VLSI Memo No. 89-529
May 1989

Sequential Design of Experiments with Physically Based Models

Michele E. Storm

Abstract

The application of physically based models to sequential optimization was explored and the benefits measured by comparison to optimizations performed using control variable polynomial models. The optimization algorithm developed is based on the sequential use of "local" (weighted) linear regression models. A new operating point or design is recommended at the optimum of the model within the region where the model is considered valid. This region, within which extrapolations based on the model are believed to be sufficiently accurate, is defined by constraints based on the estimated predictive error of the model and the distance from the data. A new model is created after each data point is collected.

As a test case, the sequential optimizer was applied to the design of a paper helicopter for maximum time of flight. The physically based model of the paper helicopter was developed through the use of dimensional analysis, a technique which groups variables according to their dimensions. It was found that the physically based model improved the design of the helicopter more rapidly than the polynomial models. For example, in one comparison of the physical model and the linear model, the physical model reached the optimum design after four sequential designs, while the linear model hadn't reached the optimum after ten designs.

The superior performance of the optimizer with the physically based model is attributed to the model providing a truer representation of the actual air-helicopter system, resulting in a larger region where the model is valid and a better choice of direction within that region.

Microsystems Research Center
Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

Submitted to the Department of Mechanical Engineering, MIT, April, 1988 in partial fulfillment of the requirements for the Degree of Master of Science in Mechanical Engineering. This work was supported in part by the Defense Advanced Research Projects Agency under contract MDA972-88-K-0008, and the Microelectronics and Computer Technology Corporation.

Author Information

Storm, *current address*: Ford Motor Company, Ignition Engineering Dept., P.O. Box 158, Ypsilanti, MI 48197. (313) 484-8687.

Sequential Design of Experiments with Physically Based Models

by

Michele E. Storm

SUBMITTED TO THE DEPARTMENT OF
MECAHNICAL ENGINEERING IN PARTIAL FULFILLMENT
OF THE DEGREE OF MASTER OF SCIENCE IN
MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

April, 1989

Signature of Author Michele E. Storm
Department of Mechanical Engineering
April 3, 1989

Certified by _____
Emanuel Sachs
Assistant Professor, Mechanical Engineering
Thesis Supervisor

Accepted by _____
Ain A. Sonin
Chairman, Graduate Committee

SEQUENTIAL DESIGN OF EXPERIMENTS WITH PHYSICALLY BASED MODELS

by

Michele Elaine Storm

Submitted to the Department of Mechanical Engineering
on April 3, 1989 in partial fulfillment of the
requirements of the Degree of Master of Science in
Mechanical Engineering

Abstract

The application of physically based models to sequential optimization was explored and the benefits measured by comparison to optimizations performed using control variable polynomial models. The optimization algorithm developed is based on the sequential use of "local" (weighted) linear regression models. A new operating point or design is recommended at the optimum of the model within the region where the model is considered valid. This region, within which extrapolations based on the model are believed to be sufficiently accurate, is defined by constraints based on the estimated predictive error of the model and the distance from the data. A new model is created after each data point is collected.

As a test case, the sequential optimizer was applied to the design of a paper helicopter for maximum time of flight. The physically based model of the paper helicopter was developed through the use of dimensional analysis, a technique which groups variables according to their dimensions. It was found that the physically based model improved the design of the helicopter more rapidly than the polynomial models. For example, in one comparison of the physical model and the linear model, the physical model reached the optimum design after four sequential designs, while the linear model hadn't reached the optimum after ten designs.

The superior performance of the optimizer with the physically based model is attributed to the model providing a truer representation of the actual air-helicopter system, resulting in a larger region where the model is valid and a better choice of direction within that region.

Thesis Supervisor: Dr Emanuel Sachs

Title: Assistant Professor of Mechanical Engineering

CONTENTS

Abstract 1

Contents 2

List of Variables 4

1 Introduction 5

1.1 Motivation 5

1.2 Related Work 5

1.3 Physical Models 6

1.4 Current Work 6

2 Development of a Sequential Optimizer 8

2.1 Model and Notation 9

2.2 Determination of the Region in which the Model is Valid 11

2.3 Mathematical Statement of the Problem 13

3 Application of Sequential Optimizer to Paper Helicopter Design 15

3.1 Description of System 15

3.2 Physical Model 15

3.3 Application of Basic Sequential Optimizer 17

4 Results and Discussion 20

4.1 Experimental Results 20

4.2 Discussion of Model Comparison 25

4.3 Suggested Improvements to the Sequential Optimization Algorithm 26

5 Conclusions 28

Acknowledgements 29

References 30

Appendix A - Derivation of Weighted Regression 31

Appendix B - Derivation of Prediction Confidence Interval 36

Appendix C - Basic Optimization Algorithm 38

Appendix D - Computer Code 47

List of Symbols

y: the objective function to be maximized by the sequential optimizer

x: the vector of control variables - the actual variables which the experimenter changes

z: the vector of predictor variables in the regression function. The elements of **z** are a function of the elements of **x**.

Z: the design matrix whose rows are $\mathbf{z}^t$

ζ: vector of random error terms associated with the linear model

V: the matrix which contains the terms, w, that determine the relative magnitude of the error variance with respect to the standard error variance

σ^2: the standard error variance, the variance of the reference point.

w: the the weighting factor for the error variance

Δ_i: scaling constants which determine the local region by determining w

Q: matrix which contains Δ_i and is part of the weighting function

β : the vector of coefficients in the linear regression model

s^2: the estimated standard variance

Σ: the covariance matrix of **x**

μ: the mean vector of **x**

L: the length of the paper helicopter blade

b: the width of the paper helicopter wing

w_t: the total weight of the paper helicopter

w_a : the weight of the paper added to the end of the helicopter tail

w_p: the weight of the paper in the helicopter

V: the descent velocity of the helicopter

1 INTRODUCTION

1.1 Motivation

One of the fundamental goals of methods of experimental design is the realization of improvement of products and processes with a minimum of experimental effort. The work described in this thesis seeks to minimize the number of experiments needed to optimize the design of a product or operation of a process through the use of physically-based models in an algorithm based on sequential design of experiments.

1.2 Related Work

There are two primary methods of experimental design; parallel and sequential. Both methods are well known and have been written about extensively in the literature. In parallel design of experiments (DOE), a set of experiments is run and then the results are used to optimize the product. Classical DOE, such as partial and full factorial designs (Box, Hunter, and Hunter, 1978) and Box-Behnken designs (Box and Behnken,1960), as well as the orthogonal arrays popular in quality control (Taguchi, 1986) fall into this category .

In sequential DOE data are analyzed as they are collected, and used to determine the next experiment to be run. The advantage of this scheme is that it allows the the experimenter to use the information available in the collected data to tailor the remaining experiments to more effectively optimize the process. Examples of this method of experimentation are the Simplex method and the Ultramax method (Moreno, 1986).

The key issue in deciding if sequentially designed experiments are applicable is the speed with which feedback from the experiments can be obtained. For example, in agricultural experiments, which require an entire growing season to complete, parallel experiments make are the most suitable because a series of sequentially performed experiments would require a prohibitively long time to complete. On the other hand, when experiments are performed on manufacturing production lines the product is usually produced sequentially and measurements on the response of interest, such as the dimensions of a part, can be obtained almost immediately. In this case, sequentially designed experiments are most appropriate.

The concept of on-line process optimization through repeated use of designed experiments was first introduced by G. E. P. Box in "Evolutionary Operation". This idea was further developed by the Ultramax Corporation, which currently markets a computer program to perform sequential optimization. This software performs sequential optimization based on quadratic polynomials relating the response to the predictor variables. Our work builds on these sources.

1.3 Physical Models

In many engineering applications of statistical methods, including the Ultramax Method, a physical system is modeled by a polynomial. This approach assumes that a polynomial can adequately describe the relationship between the response and the predictor variables. However, the engineer often has some idea about the relationship between the response and the predictor variables, for example, in the case of thin film deposition process, it may be known that the growth rate varies inversely with the pressure (P) and proportionally to flow rate (Q). In this case a form of the model that relates the grouped variable Q/P to growth rate would be more accurate than one that related P and Q to growth rate. Using the engineers' prior knowledge of a process results in the choice of a model form which more accurately represents the true situation. Both physically based modelling and sequential DOE deal with the efficient incorporation of the available information.

1.4 Current Work

Our immediate objective is to determine how physically based models enhance the performance of sequential optimizers. Our long-term objective is to explore the optimization of manufacturing processes using dedicated sequential optimization algorithms with embedded physically based models. The type of physically based model examined in this thesis is derived from what is known in the physical sciences as dimensional analysis.

In order to achieve our goal a very basic sequential optimizer was developed. This optimizer drew on the concepts of the Ultramax Method. Our code has the flexibility to allow the use of any model which is linear in the estimated coefficients. This flexibility was needed to implement some of the models. Also, developing our own simple sequential optimizer has the advantage that we know the algorithms involved so tha the

effect of the physically based models can be more precisely evaluated. The design of a paper helicopter was used to test the ability of physical models to improve the performance of a sequential optimizer.

2 DEVELOPMENT OF A SEQUENTIAL OPTIMIZER

The sequential optimizer developed for this thesis works in the following manner. Based on the available data the optimizer suggests a set of control variables which it believes to be an improvement over the last run. The process operator then runs the process at these settings. The quantity of interest is recorded as well as the settings at which the process was run. This data is then added to the data file which the optimizer uses to determine the next point that it will suggest. The cycle of running the code to obtain a suggested new set of control variables at which to run then repeats itself (see Figure 1). In this manner the process is gradually improved.

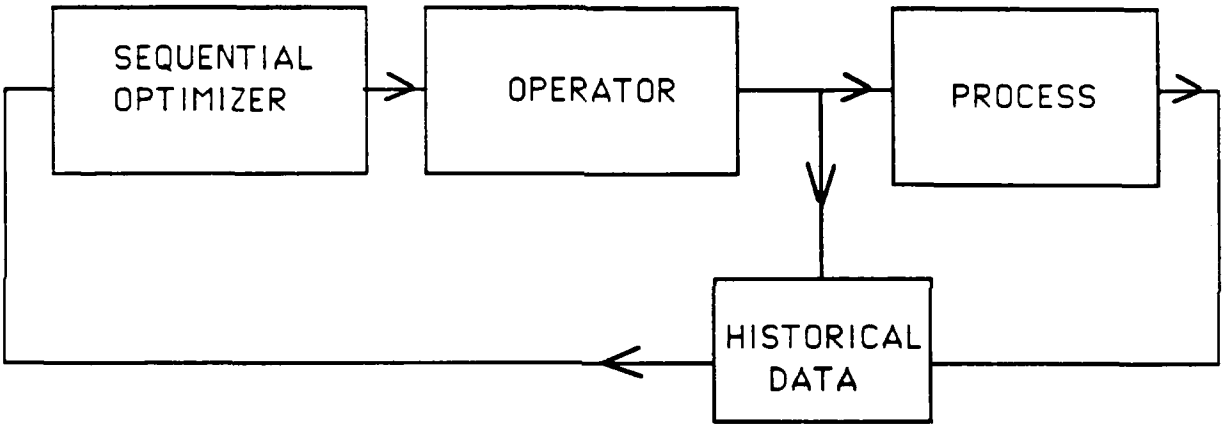

Figure 1: Diagram of process with sequential optimizer

In the code developed for this thesis there are three steps involved in determining the next recommended operating point. The first step is the creation of a model relating the input parameters to the objective function. The second step is the determination of the region in which the model is an adequate representation of the relationship between the objective function and the control variables. The last step is the optimization of the objective function based on the model created in step 1. The solution is constrained to be in the region in which the model is considered valid as determined by step 2. The optimization is also subject to the constraints that the user places on the control variables.

2.1 Model and Notation

The first basic assumption is that for each individual response the relationship between the objective function ($\mathbf{y}$) and the control variables ($\mathbf{x}$) can be locally described by an equation of the form

$$\mathbf{y} = \mathbf{Z}\beta + \zeta \tag{1}$$

where

$$\zeta \sim N(0, \mathbf{V}\sigma^2) \quad \text{and} \quad \mathbf{z} = f(\mathbf{x}) \tag{2}$$

The $\mathbf{V}$ term is defined by

$$\mathbf{V} = \begin{bmatrix} w_1 & & 0 \\ & w_2 & \\ 0 & & w_n \end{bmatrix} \tag{3}$$

where

$$w_i = \exp[(\mathbf{x}_i - \mathbf{x}_r)^t \mathbf{Q} (\mathbf{x}_i - \mathbf{x}_r)/(2m)] \tag{4}$$

and

$$\mathbf{Q} = \begin{bmatrix} 1/\Delta_1^2 & & 0 \\ & 1/\Delta_2^2 & \\ 0 & & 1/\Delta_m^2 \end{bmatrix}. \tag{5}$$

The reference point around which the model is created is denoted $\mathbf{x}_r$ and m is the number of control variables. The diagonal elements of $\mathbf{Q}$ are a function of Δ_j which is the length of the local region for the control variable x_j.

There are three important features of this formulation. The magnitude of the error, ζ_i, and hence its variance, $w_i\sigma^2 = \exp[(\mathbf{x}_i - \mathbf{x}_r)^t \mathbf{Q} (\mathbf{x}_i - \mathbf{x}_r)/(2m)]\sigma^2$, is a function of $\mathbf{x}$ and $\mathbf{Q}$. The predictor variables, $\mathbf{z}$, are a function of $\mathbf{x}$, and the model is linear in the coefficients, β.

The first important feature is the treatment of the error variance as a function of $\mathbf{x}$. In the standard formulation of the linear regression problem, the error variance is assumed to be the same for every data point. (The $\mathbf{V}$ in the present error distribution is replaced by the identity matrix $\mathbf{I}$ in the standard formulation). In our formulation, the variance of the error increases exponentially as $(\mathbf{x}_i-\mathbf{x}_r)^t\mathbf{Q}(\mathbf{x}_i-\mathbf{x}_r)$. The quantity $(\mathbf{x}_i-\mathbf{x}_r)^t\mathbf{Q}(\mathbf{x}_i-\mathbf{x}_r)$ can be seen to represent the square of the scaled distance from $\mathbf{x}_r$ by performing the matrix multiplication of the quantity

$$(\mathbf{x}_i-\mathbf{x}_r)^t\mathbf{Q}(\mathbf{x}_i-\mathbf{x}_r) \tag{6}$$

yielding:

$$(x_{i1}-x_{r1})^2/\Delta_1^{\,2} + (x_{i2}-x_{r2})^2/\Delta_2^{\,2} + \cdots + (x_{im}-x_{rm})^2/\Delta_m^{\,2}. \tag{7}$$

$\mathbf{Q}$ makes the distance dimensionless so that variables with different units can be compared. Most importantly, it defines the size and shape of the local region by determining the rate at which movement away from the reference point in a particular variable is penalized.

The use of the exponentially increasing error reflects the assumption that the model is insufficient to describe the process well over the entire range of data but that it can give an adequate description of the process in a smaller region. The model is created to be most accurate near the reference point. Inadequacies in the model are treated as if they were pure error. Although this is incorrect, it provides a useful way to treat model error. The fact that the error variance isn't constant affects the way that the coefficients β are estimated, as well as the error in the prediction of y.

Another feature of this model is that y is a function of $\mathbf{z}$, which is in turn a function of $\mathbf{x}$. The vector $\mathbf{z}$ is used to represent the transformation of the data from the original control variables. For example, if, in terms of the original control variables, the model is

$$y = \beta_0 + \beta_1\sqrt{x_1}/x_3 + \beta_2 x_2 \tag{8}$$

then

$$z_0 = 1, \quad z_1 = \sqrt{x_1}/x_3, \quad z_2 = x_2. \tag{9}$$

Although this is largely a matter of notational convenience, it helps to clarify the fact that the model of the system is more complex than a simple linear relationship between **y** and **x**. There are many choices for the relationship between **z** and **x**. One of the major focuses of this thesis is the judicious choice of transformations from **x** to **z**.

The third feature is that the model is linear in the coefficients. This means that a least squared error regression can be used to estimate the coefficients. However, because the error variance changes from one data point to another a weighted least squares regression is the appropriate regression type in this case. A weighted least squares regression minimizes:

$$\sum_i w_i^{-1}(\hat{y}_i - y_i)^2 \tag{10}$$

while an unweighted regression minimizes:

$$\sum_i (\hat{y}_i - y_i)^2. \tag{11}$$

The standard equation for a weighted least squares regression is:

$$\hat{\beta} = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{y} \tag{12}$$

The resulting estimate of β is distributed as

$$\hat{\beta} \sim N(\beta,(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}). \tag{13}$$

Brief derivations of equations (12) and (13) are given in Appendix A.

2.2 Determination of the Region in which the Model is Valid

In order to determine the region in which the model will be considered valid, two criteria were used. The first is based on the prediction error of y. The second criteria is based on the "distance" from the data that created the model. The first criterion is statistically based. If the assumptions about the form of the model and the nature of the variance are correct, then the 90% confidence interval for the prediction value is given by:

$$y_0 = \hat{y}_0 \pm t_{.05}\sqrt{z_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}z_0 s^2 + w_0 s^2} \tag{14}$$

or

$$y_0 = \hat{y}_0 \pm t_{.05}\sqrt{z_0^t(Z^tV^{-1}Z)^{-1}z_0 s^2 + \exp[(x - x_r)^tQ(x - x_r)/(2m)]s^2}\,. \qquad (15)$$

The first term under the radical arises from the error in the estimate $\hat{\beta}$ which propagates into $\hat{y}_0$ through $z_0^t\hat{\beta}$, while the second term comes from the variance of the error at the point x_0.

This formulation differs somewhat from the standard prediction confidence interval,

$$y_0 = \hat{y}_0 \pm t_{.05}\sqrt{z_0^t(Z^tZ)^{-1}z_0 s^2 + s^2}\,, \qquad (16)$$

due to the fact that the variance of $\hat{\beta}$ is $(Z^tV^{-1}Z)^{-1}\sigma^2$ instead of the usual $(Z^tZ)^{-1}\sigma^2$ and the variance of ε_0 is $w_0\sigma^2 = \exp[(x_0-x_r)^tQ(x_0-x_r)/(2m)]\sigma^2$ instead of σ^2[1]. The prediction confidence limit of equation (15) is derived in Appendix B.

The model is considered valid within the region where half of the 90% confidence interval is less than some value (K1) determined by the user. The magnitude of this value can be a reflection of the risk of a poor product that the user is willing to take. This constraint becomes:

$$t_{.05}\sqrt{z_0^t(Z^tV^{-1}Z)^{-1}z_0 s^2 + \exp[(x - x_r)^tQ(x - x_r)/(2m)]s^2} \le K_1 \qquad (17)$$

The second criteria concerning the distance form the data is a rule of thumb. In general it is unwise to extrapolate far from the data that created the model. Even though a model may appear to fit the data well, this does not necessarily mean that the model will still be good far from the data from which it was created.

This constraint reflects the fact that we do not know the true form of the equation relating the control variables to the output. If the form of the equation were known then

---

1 We chose to use the prediction of y confidence interval instead of the more common expected value of y confidence interval, which for our model would be $y_0 = \hat{y}_0 \pm t_{.05}\sqrt{z_0^t(Z^tV^{-1}Z)^{-1}z_0 s^2}$ because the prediction confidence interval is a function of the error at the prediction point, and we are using the error variance to "explain" the lack of fit of the model away from the reference point.

this constraint would be unnecessary. To quantify the feeling that we should not extrapolate far from the data, the generalized multivariate squared distance is employed. The multivariate squared distance is:

$$(\mathbf{x} - \mu)^t \Sigma^{-1} (\mathbf{x} - \mu) \tag{18}$$

where Σ is the covariance matrix of $\mathbf{x}$. This is the multivariate analog of the univariate squared distance:

$$[(x-\mu)/\sigma]^2 = (x-\mu)^t \sigma^{-2} (x-\mu) \tag{19}$$

These equations measure distance from the population mean in terms of standard deviations (Johnson and Wichern, 1988). The user then determines the maximum (K2) that this squared distance can be. Again the choice of this constant can be a reflection of the risk that the user is willing to take. If $\mathbf{x}$ were multivariate normally distributed then multivariate squared distance would have a chi-squared distribution. $\mathbf{X}$ is not multivariate normally distributed, however in order to develop a measurement of how far away is too far, we chose to use $2\chi^2_{0.10}$ to define the limits of the region in which an extrapolation would be considered good. Since we do not know Σ we replace it in equation (18) by its estimate $\mathbf{S}$. Likewise $\hat{\mu}$ is substituted for μ. We arrive at the second constraint :

$$(\mathbf{x} - \hat{\mu})^t \mathbf{S}^{-1} (\mathbf{x} - \hat{\mu}) \leq K2; \text{ where } K2 = 2\chi^2_{0.10}. \tag{20}$$

These two constraints determine the region in which the model of the process is considered valid.

2.3 Mathematical Statement of the Problem

We may now state the problem as:

$$\text{Maximize } \hat{y} = z^t \hat{\beta}$$

subject to:

$$t_{.05} \sqrt{z_0^t (Z^t V^{-1} Z)^{-1} z_0 s^2 + \exp[(\mathbf{x} - \mathbf{x}_r)^t \mathbf{Q} (\mathbf{x} - \mathbf{x}_r)/(2m)] s^2} \leq K_1 \tag{21}$$

$$(\mathbf{x} - \hat{\mu})^t \mathbf{S}^{-1} (\mathbf{x} - \hat{\mu}) \leq 2\chi^2_{0.10}$$

and constraints imposed by the user, such as

$$x_2 \geq 3$$
$$x_1 + x_3 \leq 7$$

A nonlinear optimizer solves this problem. The solution is the next point recommended by the sequential optimization program. The nonlinear optimization algorithm is discussed in Appendix C.

3 APPLICATION OF SEQUENTIAL OPTIMIZER TO PAPER HELICOPTER DESIGN

For the purpose of testing the value of using physically based models in a sequential optimization algorithm, the sequential optimizer was applied to the design of a paper helicopter.

3.1 Description of System

When a paper helicopter of the type shown in Figure 2 is dropped, it begins to spin as it falls downward. The spinning breaks the free-fall of the helicopter and prolongs the time the helicopter remains aloft. The three design parameters which affect flight time are the length of the blade (L), the width of the wing (b), and the total weight (w_t) of the helicopter. The total weight is equal to the weight of the paper (w_p) plus the weight added (w_a) at the end of the tail. The angle at which the blades are initially bent was determined to have negligible affect during early screening experiments. We chose to design the helicopters for long flight time or, equivalently, low velocity (V). (Actually, negative velocity was used as the objective function since the simple optimizer was set up for maximization.)

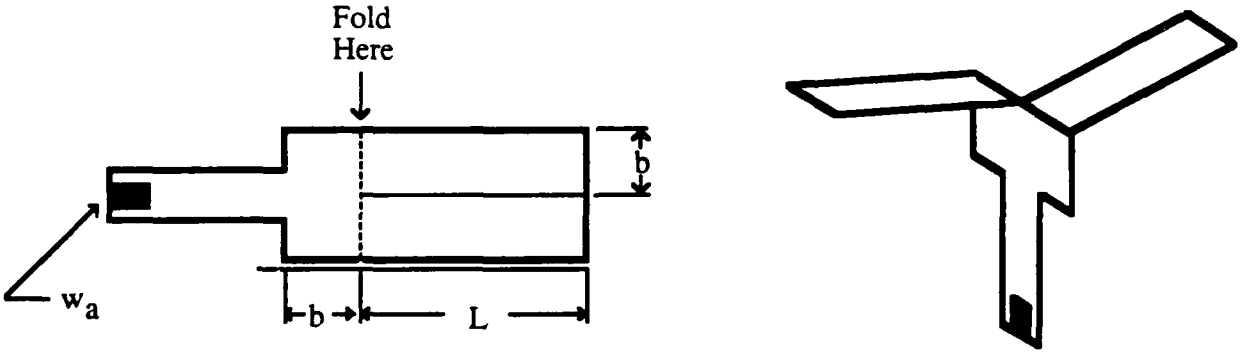

Figure 2: Helicopter design

3.2 Physical Model

Dimensional analysis is a technique commonly used in the physical sciences to simplify the analysis of complex multivariable problems by grouping variables into

dimensionless quantities. The underlying basis for dimensional analysis is that physical results must be independent of the system of units chosen for their measurement.

The Buckingham Pi theorem (Bridgman, 1931) provides a step by step approach to the formation of sets of dimensionless quantities which are appropriate to a specific problem. The set of dimensionless variables is smaller in number that the original set of variables. In the case of the helicopter, there are four variables which influence the velocity of the helicopter; L, b, w, and ρ, where ρ is the density of the air. We may formulate an implicit relationship between V and the four variables as follows:

$$g(V, L, b, w, \rho) = 0 \tag{22}$$

An application of the Pi Theorem results in the transformation of these five variables into two dimensionless groupings $V\sqrt{\rho}\,b/\sqrt{w}$, and L/b, from which an implicit relationship may be formed which characterizes the helicopter problem:

$$g(V\sqrt{\rho}\ b/\sqrt{w}, L/b) = 0 \tag{23}$$

This relationship may be restated as:

$$V\sqrt{\rho}\ b/\sqrt{w} = f(L/b) \tag{24}$$

We chose to approximate the function of (L/b) with

$$f(L/b) = \beta_0 + \beta_1 L/b + \beta_2 (L/b)^2 \tag{25}$$

so that

$$V\sqrt{\rho}\ b/\sqrt{w} = \beta_0 + \beta_1 L/b + \beta_2 (L/b)^2. \tag{26}$$

Since ρ is constant it will be absorbed into the coefficients. The model then becomes

$$V = \sqrt{w}/b[\ \beta_0 + \beta_1 L/b + \beta_2 (L/b)^2]. \tag{27}$$

The application of dimensional analysis to the helicopter problem has provided benefit in two ways. First, the number of experimental control factors has been reduced

from the original list of L,b, and w to $V\sqrt{\rho}b/\sqrt{w}$ and L/b. Second, the grouping of the variables into the dimensionless terms and the grouping of the terms themselves stems from physical considerations, and may therefore be expected to model the process more accurately than a simple polynomial model.

A by-product of the fact that the number of experimental parameters has been reduced from 3 to 2 is that specification of the two dimensionless parameters $V\sqrt{\rho}b/\sqrt{w}$ and L/b does not uniquely specify the three helicopter design parameters L, b, and w. Indeed, if the model resulting from dimensional analysis captured all physical effects (which it does not), the two dimensionless parameters would be sufficient to predict the performance of the design. Since the the dimensional analysis is an approximate physical model, an additional term, L, is added to the model so that L, b, and w are independently determined. The final physically based model is:

$$V = \sqrt{w}/b[\ \beta_0 + \beta_1 L/b + \beta_2 (L/b)^2] + \beta_3 L. \tag{28}$$

3.2 Application of Basic Sequential Optimizer

The sequential optimizer was used to improve the design of the helicopters. The optimization process was performed with the three different models shown below:

$$\text{Model 1: } V = \sqrt{w}/b[\ \beta_0 + \beta_1 L/b + \beta_2 (L/b)^2] + \beta_3 L \tag{29}$$

$$\text{Model 2: } V = \beta_0 + \beta_1 L + \beta_2 b + \beta_3 w \tag{30}$$

$$\text{Model 3: } V = \beta_0 + \beta_1 L + \beta_2 b + \beta_3 w + \beta_4 L^2 + \beta_5 b^2 + \beta_6 w^2 + \beta_7 Lb + \beta_8 Lw + \beta_9 bw \tag{31}$$

The first model is the physical model based on the dimensional analysis. The second model is a model which is linear in the basic variables L, b, and w. Model 1 and Model 2 both have four terms. The third model is a full quadratic in the basic variables.

In order to start the optimization process, data sufficient to estimate the coefficients and the error variance of the models were required. The starting data were obtained through parallel DOE. The sequential optimizer was run using two separate sets of starting data, Set 1, and Set 2, shown in Table 1 and Table 2, respectively. This was done so that the

effect of the models on the optimization speed could be compared under different starting conditions. Model 1 and Model 2 were compared based on the first set of starting data. The first set of starting data came from a two level full factorial DOE.

Set 1

RUN #	L (cm)	b (cm)	w_t (g)	V ave (cm/s)
1	8.0	2.5	10.0	338.0
2	12.0	2.5	10.0	187.8
3	8.0	3.5	10.0	265.5
4	12.0	3.5	10.0	181.9
5	8.0	2.5	12.0	390.3
6	12.0	2.5	12.0	201.8
7	8.0	3.5	12.0	313.8
8	12.0	3.5	12.0	184.0

Table 1: First set of starting data

All three models were then compared based on the second set of starting data. Since there were initially only six data points the full third model could not be estimated. Due to this lack of data points, only the first four terms of the model (coefficients 0 - 3) were used at the first step, and an additional term from the full quadratic model was added after each additional experiment.

Set 2

RUN #	L (cm)	b (cm)	w_t (g)	V ave (cm/s)
1	9.0	2.0	10.0	344.9
2	7.0	3.0	10.0	329.8
3	9.0	3.0	12.0	290.0
4	8.0	2.5	11.0	356.5
5	8.0	3.0	11.0	303.4
6	9.0	3.0	10.0	260.3

Table 2: Second set of starting data

For this problem there were two user constraints on the control variables. Since the paper from which the helicopters were constructed was 25 cm long, the length of the

helicopter, excluding the tail, was constrained to be less than 25 cm. Also, the lower bound on helicopter weight was set at 8 grams. This weight bound was chosen so that the optimizer would never recommend a physically impossible design - one where the weight of the paper, w_p, in the helicopter which is determined by L and b would be greater than the recommended weight, w_t. The constrained optimization problem was

$$\begin{aligned} &\text{Maximize } (-V) \\ &\text{subject to:} \qquad (32) \\ &L + b \leq 25 \\ &w \geq 8 \end{aligned}$$

The user specified parameters were set as follows. The Δ_i, the scaling constants of the error weighting function of equation (4),were chosen to be the range of their respective control variables x_i in the initial data set. This means that for the the experiments which began with the first initial data set, $\Delta_1 = 4$, $\Delta_2 = 1$, and $\Delta_3 = 2$. For the second initial data set $\Delta_1 = 2$, $\Delta_2 = 1$, $\Delta_3 = 2$. K1, the variable which determines the maximum error in the prediction of y as specified in equation (17), is also set by the user. For the design processes based on the first set of initial data, K1 was set at 67.1 for the entire run of the optimization. For the design processes based on the second set of initial data, K1 was originally set at 23.5 cm/s. However, after the first 2 sequential experiments, K1 had to be changed to 67.1 cm/s due to the high predictive uncertainty in Models 2 and 3. K2, the bound for the distance constraint, was set at $2\chi^2_{.01} = 12.5$ for all of the optimizer applications.

Data on the descent velocity were obtained by releasing the helicopters in a stairwell from a height of three stories and the time of flight was used to calculate velocity. Because of variability in the flight time, replicates of each experiment were performed and averaged to become one data point in the optimization. The "velocity" entered into the sequential optimizer was calculated as 100/time, however the actual velocity (in terms of cm/s) is recorded in the tables of this thesis. An occasional clearly outlying point was not considered in the determining the average velocity. After the starting data was collected, the sequential optimizer was then used to determine the subsequent helicopter designs.

4 RESULTS AND DISCUSSION

4.1 Experimental Results

The optimizer was run with five model and initial data set combinations, specifically: Model 1 and Set 1, Model 2 and Set 1, Model 1 and Set 2, Model 2 and Set 2, and Model 3 and Set 2. The presentation of the data is organized according to the initial data set.

The results of the comparison of Model 1 and Model 2 based on the Set 1 of starting data are presented in Figure 3. The first eight data points are from the initial data set obtained through parallel design, and the remaining data points are the result of the sequential optimizations.

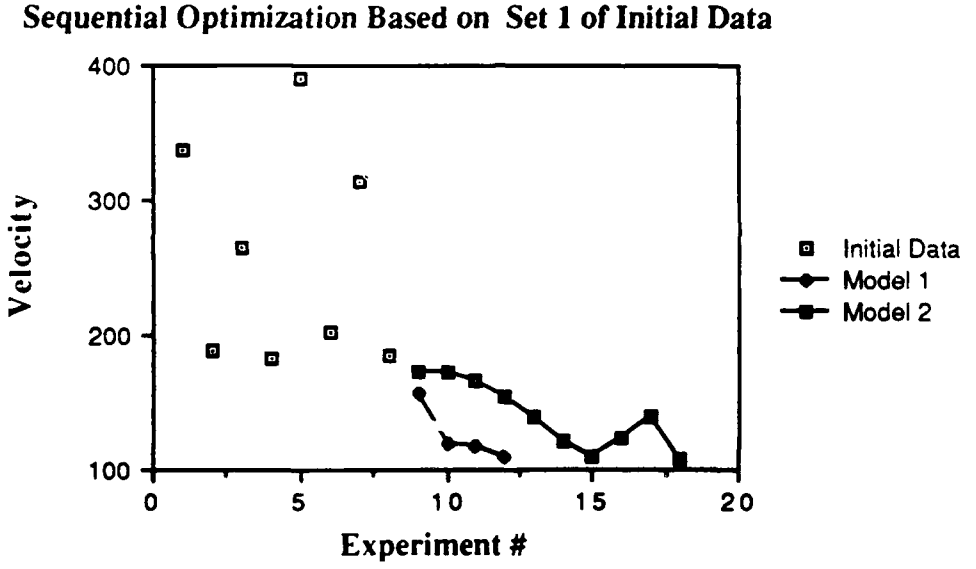

Figure 3: Optimizations based on the first set of initial data

As may be seen by referring to Figure 3, the performance of the optimizer with the dimensional analysis model (Model 1) was found to be superior to the performance of the optimizer with the linear model. The optimizer based on the dimensional analysis model reached a point near the optimum by the fourth sequential design. At this point the next recommended design differed only slightly from the last and we chose to treat this design as the optimum. The optimization based on the linear model had not yet reached the optimum design by the tenth sequential design, although it had also improved the helicopter design.

The step by step evolution of the helicopter designs in L-b design space illustrates the way in which the sequential optimizer functions. The location of the designs for the optimizations based on Set 1 are shown in Figures 4 and 5. The shaded region represents the infeasible designs, where L + b > 25.

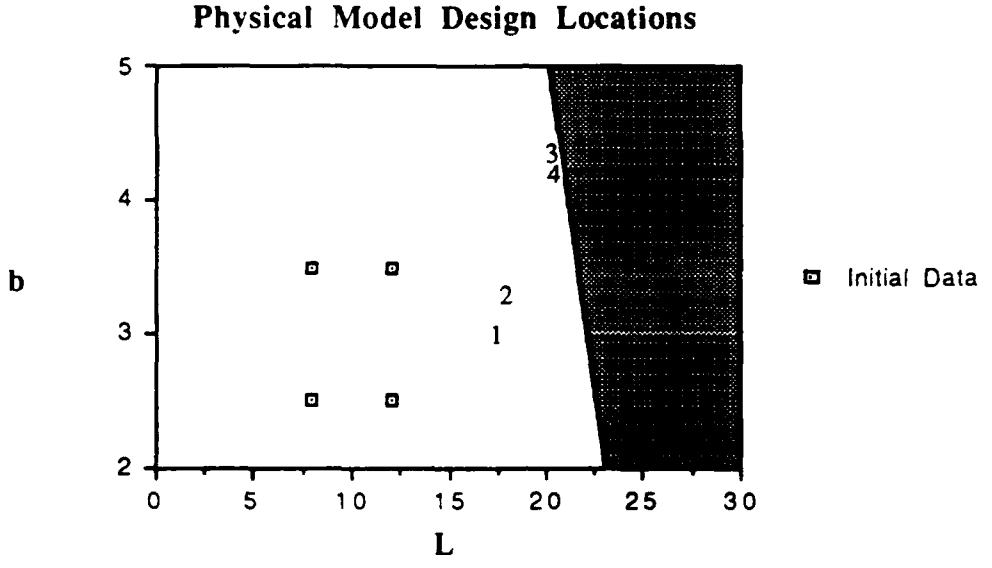

Figure 4: Design locations for the optimization based on Model 1.

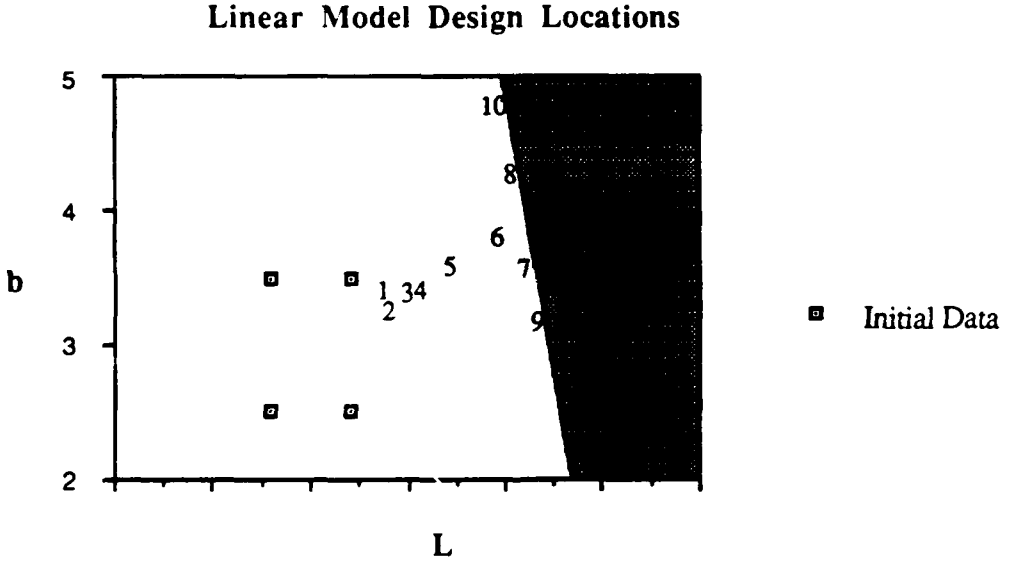

Figure 5: Design locations for the optimization based on Model 2.

The optimum design appears to be on the bounds of the design region - where L + b =25, and w = 8.

The actual values of the data plotted in Figures 4 and 5 are listed in Tables 3 and 4. Additionally, the predicted value of velocity , as well as half of the width of the 90% confidence interval for the prediction are given. Note that the prediction error constraint of equation (17) requires that half of the width of the 90% confidence interval be less than K1, which is 67.1 cm/s for the design processes based on the first set of initial data. If the optimum for the step was on the bound of the region where the model is considered valid, then the constraint which caused the bound is listed in the constr column. The letter p indicates that the prediction error constraint, which is defined by equation (17) and the choice of K1, was the binding constraint, while d indicates that the distance constraint, defined by equation (20) and the choice of K2, was the binding constraint.

Physical Dimensional Anaysis Model

RUN #	L (cm)	b (cm)	w_t (g)	con-straint	V ave (cm/s)	V pred (cm/s)	± (cm/s)
1	17.4	3.0	9.9	d,p	157.7	13.2	67.1
2	18.0	3.3	8.0	p	120.0	90.3	67.1
3	20.6	4.4	8.0	d	116.8	65.4	66.8
4	20.7	4.3	8.0	*	110.6	109.2	29.5

Table 3: Results of the sequential optimization based on Model 1 and the first set of initial data

Linear Model

RUN #	L (cm)	b (cm)	w_t (g)	con-straint	V ave (cm/s)	V pred (cm/s)	± (cm/s)
1	13.7	3.4	10.7	p	171.6	113.9	67.1
2	13.8	3.3	10.5	p	171.6	137.3	67.1
3	15.0	3.4	10.4	p	166.2	110.8	67.1
4	15.5	3.4	10.4	p	155.6	118.1	67.1
5	17.0	3.6	10.2	p	138.9	93.1	67.1
6	19.5	3.8	10.0	p	120.7	61.8	67.1
7	21.4	3.6	8.7	p	109.2	65.4	67.1
8	20.7	4.3	11.6	p	123.2	80.3	67.1
9	21.8	3.2	8.0	p	138.8	83.4	67.1
10	20.2	4.8	11.4	p	107.8	91.1	67.1

Table 4: Results of the sequential optimization based on the Model 2 and the first set of initial data

Run eight was repeated for the linear model and the second data point was used in the sequential optimization process.

The next group of results comes from the applications of the sequential optimizer based on the second set of starting data. All three models were compared based on this set of starting data. Figure 6 shows the results of all three design processes. The first six data points are from the initial data set.

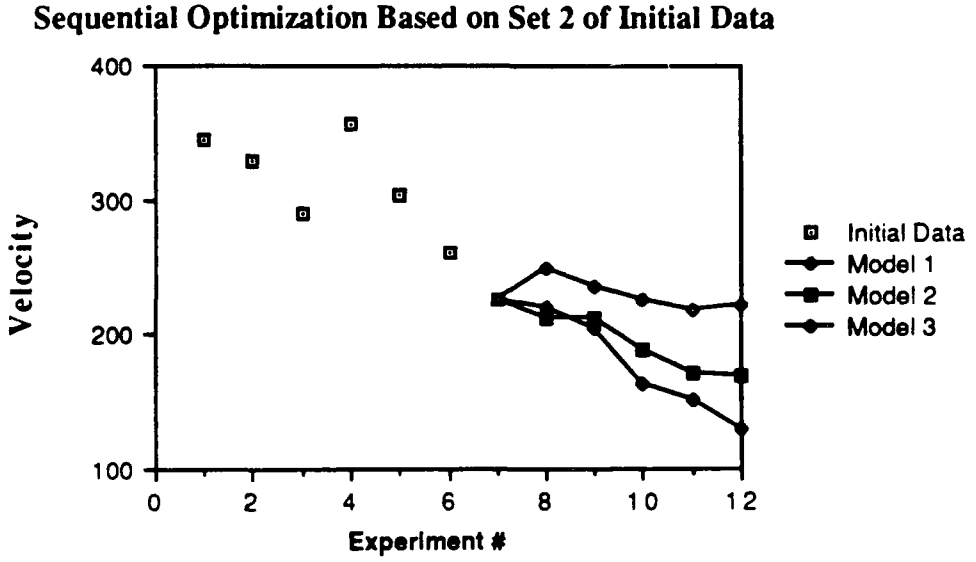

Figure 6: Optimizations based on the second set of initial data

The comparison of Model 1 and Model 2 based on the second set of initial data gives results similar to those based on the first initial data set. The physically based model improves the design the most, although both models appears to improve the helicopter design.The third model, which adds a new term from the full quadratic at each step, gave the weakest performance, although it too appeared to improve the design. None of the models reached a final design by the end of the six sequential designs.

The numerical results of the three design processes are listed in tables 4-6.

Physical Dimensional Analysis Model

RUN #	L (cm)	b (cm)	w_t (g)	con-straint	V ave (cm/s)	V pred (cm/s)	± (cm/s)
1	8.6	3.0	8.5	p	225.7	293.3	23.5
2	8.9	3.1	8.0	p	219.3	204.9	23.5
3	10.2	3.6	8.0	d	204.9	149.4	30.3
4	11.4	4.0	8.0	d	162.8	157.9	48.5
5	12.8	4.5	8.0	d	151.7	130.5	35.0
6	14.6	5.1	8.0	d	129.8	121.1	25.6

Table 5: Results of the sequential optimization based on the Model 1 and the second set of initial data

Linear Model

RUN #	L (cm)	b (cm)	w_t (g)	con-straint	V ave (cm/s)	V pred (cm/s)	± (cm/s)
1	9.3	3.5	10.7	p	225.3	214.7	23.5
2	10.2	4.2	10.9	d	212.7	137.3	22.5
3	11.1	4.8	10.9	p	212.3	141.5	67.1
4	11.9	5.4	10.9	p	118.3	158.2	67.1
5	14.1	7.1	11.1	d	169.7	107.8	66.3
6	16.4	8.2	10.7	d,p	167.8	119.7	67.0

Table 6: Results of the sequential optimization based on the Model 2 and the second set of initial data

Quadratic Model

RUN #	L (cm)	b (cm)	w_t (g)	con-straint	V ave (cm/s)	V pred (cm/s)	± (cm/s)
1	9.3	3.5	10.7	p	225.3	214.7	23.5
2	9.3	3.7	10.7	p	249.7	204.9	23.5
3	9.6	3.7	9.6	p	235.2	206.3	67.1
4	9.8	3.6	9.8	p	224.6	191.9	67.1
5	10.0	3.4	9.8	p	218.1	203.0	67.1
6	10.3	3.3	10.0	p	221.6	200.5	67.1

Table 7: Results of the sequential optimization based on the Model 3 and the second set of initial data

The distance constraint frequently limited the physical model while the polynomial models were limited by the constraint that the prediction confidence interval be less than 23.5 for the first two sequential designs and less than 67.1 for the remaining sequential designs. Run number 5 was repeated for both the linear and quadratic models. The sequential optimization process proceeded on the basis of the repeated runs.

4.2 Discussion of Model Comparison

The physically based dimensional analysis model performed the best for both sets of initial data. The improvement in the helicopter design after a given number of steps was greater for the optimization based on the dimensional analysis model than it was for the optimization based on either of the polynomial models. This can be attributed to two interconnected effects which reflect the fact that the physical model is a better representation of the true system. The first is that the physically based model provided a better fit to the data so that the region in which the model is considered valid was larger. The second is that the model provides for better direction within the region. This region, where both the distance constraint and the prediction error constraint are satisfied, is larger, because the prediction error constraint is less limiting. (The distance constraint, which is a function of the distribution of the control variables, is independent of the model, while the prediction error constraint depends on how well the model fits the data.) Extrapolations based on the model are believed to be sufficiently accurate within the region where the model is valid. The sequential optimizer does recommend designs beyond this point. Because of the larger region, the design could change more between each step, speeding the sequential optimization process.

The physical model frequently placed the step optimum on the bound created by the distance constraint, not the prediction error, indicating that poor fit of the model did not limit the optimization. The step optimum was on the distance bound for 6 of 10 steps for the physical model. (The step optimum was at the intersection of the distance and prediction error constraints at one of these points.) This was in part due to the good fit of the physical model to the data. Another factor is that if the optimum was at the distance bound at the previous step it may have the propensity to be there again since the distance constraint may not change significantly between steps, with the addition of one data point. The distance is measured in standard deviations of the data, and the standard deviation may not change much with the addition of one point. The polynomial models, on the other hand, rarely reached the distance constraint, being limited by the prediction error of the model. The step optimum for the polynomial models was on the distance bound only 3 of 22 times and one of these points was at the intersection of the prediction error ond the distance constraints.

Although the simple linear model performs well for the sequential optimization of the paper helicopter, linear models in general will not be able to optimize processes or designs where the relationship between the response and the basic variables contains curvature. A quadratic model would resolve this problem.

The quadratic model, which contains the linear model as a subset of its terms, performs poorly due to its high predictive error, which restricts the region in which the model is valid. This is due to the large number of estimated coefficients. Each estimated coefficient has a variance as well as a covariance with the each of the other estimated coefficients. The variance and covariance of each estimated coefficient propagates into the variance of y.

4.3 Suggested Improvements to the Sequential Optimization Algorithm

The optimization algorithm performed well, improving the design of the paper helicopter for both starting data sets and all three models. However, the algorithm can be improved in a number of areas.

One aspect of the current algorithm which is unsatisfactory is that if the model is a poor fit to the data and the prediction error is high, then there may be not be a region where

the model is valid. The optimizer cannot function in this situation unless the bound on the maximum prediction error (K1) is increased, meaning that less accurate predictions are acceptable. This happened during the optimization processes based on Model 2 and 3 and the starting data from Set 2. K1 was originally set at 23.5, but then had to be changed to a larger value after two sequential designs. Though this was not critical in the helicopter application of this paper, in many on-line applications where avoiding the production of scrap is a primary concern, the accuracy of the model predictions is important. One solution, exemplified by the physically based model, is the usage of better models.

Another solution for the case where a better model is not available, is the shrinkage of the local region, so that the region where a good fit to the data is required is smaller, since the problem results from the model being insufficient to explain the data over the entire weighted range. Linear and quadratic models, in particular will be accurate over a sufficiently small region. This could be accomplished by decreasing one or all of the Δ_i's. However, this may result in models created from few effective points. This problem could be resolved by adding an additional constraint that the model be based on a minimum number of effective data points. When this constraint is violated, the number of effective points could be increased by doing experiments which are close to the reference point.

Additional suggestions for improving the algorithm include using the best data point instead of the last data point as the reference point around which the model is created. A further improvement would be the development of a starting algorithm which would eliminate the need for a parallel design of starting data. The basic idea of a starting algorithm would be to begin optimization process while the initial set of data is being collected.

5 CONCLUSIONS

In this work we have sought to explore the benefits of incorporating physically based models into a sequential optimization algorithm. We hypothesized that physically based models which incorporate an engineer's prior knowledge of a system would lead to more rapid optimization of products and processes.

In order to test this idea, we developed a simple sequential optimizer which could be modified to accommodate different models. The optimization algorithm is based on the sequential use of local models created through weighted linear regression. A new operating point or design is recommended at the optimum of the model within the region where the model is considered valid. Extrapolations based on the model are believed to be accurate within this region. A new model is created after each new data point is collected.

As a test case, we decided to design a paper helicopter for maximum flight time. The optimizer was run using a physically based model and simple polynomial models in order to compare them. The physically based model was derived by dimensional analysis, a technique that groups variables based on their dimensions. The use of the physically based model derived from dimensional analysis significantly reduced the number of steps required to optimize the helicopter design.

Our work indicates that the use of physically based models in the context of sequential optimization leads to the rapid optimization of processes and product designs. We feel that this approach is especially useful for the optimization of manufacturing processes. Future work might include the development and verification of physically based models for important classes of processes. One can envision equipment controllers that have embedded in them optimization capabilities, each with a process specific model. The capability to rapidly optimize processes will become more valuable as the trend toward flexible manufacturing with its shorter runs and change over times continues.

ACKNOWLEDGEMENTS

The author would like to thank Dr. Carlos Moreno and Mr. Peter Paterson of the Ultramax Corporation, with out whose help this thesis would not have been possible. The Ultramax software was an important motivation for the work in this thesis. The author would like to thank Dr. Emanuel Sachs, the thesis supervisor, for his guidance, encouragement, and enthusiasm that made this research effort a learning experience and a worthwhile effort. The author would also like to extend appreciation to fellow research assistants, Sungdo Ha and Ke-Jeng Hu, for providing valuable assistance in the experimental phase of this work. In the programming part of this work, many members of the Electrical Engineering community shared their computer knowledge, particularly Jarvis Jacobs. Research Assistants William Wehrle and Parmeet Chaddha provided helpful suggestions.

Special thanks go to the Defense Advanced Research Projects Agency under contract MDA972-88-K-0008 as well as the Microelectronics and Computer Technology Corporation for the support of this project .

Also, important to the completion of this effort was the moral support given by the author's parents and friends, especially Suguna Pappu, Debbie Stephan, Helen Han, Janet Pan, Joanne Hetzler, Kim Adams and Jim Green.

REFERENCES

Bridgman, P. W.; 1931, Dimensional Analysis, Yale University Press; New Haven, Connecticut

Box, G. E. P. and Behnken, D. W.; 1960, Some New Three Level Designs for the Study of Quantitative Variables, Technometrics, Volume 2, 455-475

Box, G. E. P.; Draper, N. R.; 1969, Evolutionary Operation, John Wiley & Sons, New York

Box, G. E.P., Hunter, W. G.; and Hunter, J. S.; 1978, Statistics for Experimenters, John Wiley & Sons, New York

Draper, N. R.and H. Smith; 1981, Applied Regression Analysis, John Wiley & Sons, New York

Johnson, R. A. and Wichern, D. W.; 1988, Applied Mutivariate Statistical Analysis, Prentice Hall, Englewood Cliffs, New Jersey

Luenberger, D. G., 1984, Linear and Nonlinear Programming, Addison-Wesley Publishing; Reading, Massachusetts

Moreno, Carlos W; 1986, Self-Learning Optimizing Control Software, Proceedings ROBEXS '86, from Instrument Society of America Proceedings, 371-377

Reklaitis, G. V.; Ravindran, A and Ragsdell, K. M.; 1983, Enginerering Optimization: Methods and Applications, John Wiley & Sons, New York

Taguchi, G.; 1986, Introduction to Quality Engineering, Asian Productivity Organization, Tokyo

APPENDIX A

DERIVATION OF WEIGHTED REGRESSION

A brief derivation of the formula for weighted regression is given in this appendix. For a more thorough discussion see Draper and Smith.

Weighted regression is used when the error at each observation does not (or is assumed to not) conform to the standard linear regression assumption that the error variance of each observation is equal and independent. Weighted regression basically consists of transforming the variables involved in the regression so that these assumptions are satisfied. This is done by premultiplying the standard linear regression model by a matrix so that the resulting error conforms to the standard regression assumptions.

In the context of least squared regression it amounts to penalizing the squared error differently for different points. The model of the situation where weighted regression is used is:

$$y = Z\beta + \zeta \tag{A.1}$$

where

$$\zeta \sim N(0, V\sigma^2) \tag{A.2}$$

Since **V** is square and symmetric **P** can calculated such that

$$\mathbf{P}^t\mathbf{P} = \mathbf{P}\mathbf{P} = \mathbf{P}^2 = \mathbf{V}. \tag{A.3}$$

Premultiplying the linear model by $\mathbf{P}^{-1}$ yields:

$$\mathbf{P}^{-1}y = \mathbf{P}^{-1}\mathbf{Z}\beta + \mathbf{P}^{-1}\zeta. \tag{A.4}$$

Let the following variables be defined by

$$y^* = \mathbf{P}^{-1}y, \quad \mathbf{Z}^* = \mathbf{P}^{-1}\mathbf{Z}, \text{ and } \zeta^* = \mathbf{P}^{-1}\zeta \tag{A.5}$$

Thus equation (A.4) can be rewritten as

$$y^* = Z^*\beta + \zeta^*. \tag{A.6}$$

where the distribution of ζ^* is

$$\zeta^* \sim N(0, I\sigma^2) \tag{A.7}$$

(This is because $Cov(\mathbf{Cx}) = \mathbf{C}Cov(\mathbf{x})\mathbf{C}^t$ when $\mathbf{C}$ is a constant matrix and $\mathbf{x}$ is a random variable vector.) The standard regression formula can now be applied. The standard regression equation is

$$\hat{\beta} = (\mathbf{Z}^t\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{y} \tag{A.8}$$

By substituting the starred quantities into this equation we arrive at

$$\hat{\beta} = (\mathbf{Z}^{*t}\mathbf{Z}^*)^{-1}\mathbf{Z}^{*t}\mathbf{y}^* \tag{A.9}$$

Returning to the original variables, the equation is

$$\hat{\beta} = [(\mathbf{P}^{-1}\mathbf{Z})^t(\mathbf{P}^{-1}\mathbf{Z})]^{-1}(\mathbf{P}^{-1}\mathbf{Z})^t(\mathbf{P}^{-1}\mathbf{y}). \tag{A.10}$$

Since $(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t$ the expression can be rewritten as

$$\hat{\beta} = [\mathbf{Z}^t(\mathbf{P}^{-1})^t(\mathbf{P}^{-1}\mathbf{Z})]^{-1}\mathbf{Z}^t(\mathbf{P}^{-1})^t(\mathbf{P}^{-1}\mathbf{y}) \tag{A.11}$$

After applying the fact that $(\mathbf{A}^t)^{-1} = (\mathbf{A}^{-1})^t$ the equation becomes

$$\hat{\beta} = [\mathbf{Z}^t(\mathbf{P}^t)^{-1}(\mathbf{P}^{-1}\mathbf{Z})]^{-1}\mathbf{Z}^t(\mathbf{P}^t)^{-1}(\mathbf{P}^{-1}\mathbf{y}) \tag{A.12}$$

Since $\mathbf{P}^t = \mathbf{P}$

$$\hat{\beta} = [\mathbf{Z}^t\mathbf{P}^{-1}(\mathbf{P}^{-1}\mathbf{Z})]^{-1}\mathbf{Z}^t\mathbf{P}^{-1}(\mathbf{P}^{-1}\mathbf{y}) \tag{A.13}$$

which reduces to

$$\hat{\beta} = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{y} \qquad (A.14)$$

The covariance of $\hat{\beta}$ is required for the calculation of confidence intervals. The covariance is derived as follows:

$$\mathrm{Cov}(\hat{\beta}) = \mathrm{Cov}[(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{y}] \qquad (A.15)$$

The first step is applying the relationship $\mathrm{Cov}(\mathbf{Cx}) = \mathbf{C}\mathrm{Cov}(\mathbf{x})\mathbf{C}^t$ to equation (A.15). This results in

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{y})[(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}]^t \qquad (A.16)$$

The expression of equation (A.16) can be simplified. The next 3 steps are basic matrix manipulation applying $(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t$ and $(\mathbf{A}^t)^{-1} = (\mathbf{A}^{-1})^t$ to equation (A.16).

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{y})(\mathbf{V}^{-1})^t\mathbf{Z}[(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^t]^{-1} \qquad (A.17a)$$

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{y})(\mathbf{V}^t)^{-1}\mathbf{Z}[\mathbf{Z}^t(\mathbf{V}^{-1})^t\mathbf{Z}]^{-1} \qquad (A.17b)$$

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{y})(\mathbf{V}^t)^{-1}\mathbf{Z}[\mathbf{Z}^t(\mathbf{V}^t)^{-1}\mathbf{Z}]^{-1} \qquad (A.17c)$$

finally, since $\mathbf{V}^t = \mathbf{V}$, the equation becomes:

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{y})\mathbf{V}^{-1}\mathbf{Z}(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1} \qquad (A.18)$$

Substituting $\mathbf{y} = \mathbf{Z}\beta + \zeta$ into equation (A.18) yields

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\mathbf{Z}\beta + \zeta)\mathbf{V}^{-1}\mathbf{Z}(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1} \qquad (A.19)$$

Since $\mathbf{Z}\beta$ is constant

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathrm{Cov}(\zeta)\mathbf{V}^{-1}\mathbf{Z}(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}. \quad \text{(A.20)}$$

Substituting $\mathrm{Cov}(\zeta) = \mathbf{V}\sigma^2$ into equation (A.20) yields

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{V}\sigma^2\mathbf{V}^{-1}\mathbf{Z}(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1} \quad \text{(A.21)}$$

which can be rearranged as

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z}(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\sigma^2 \quad \text{(A.22)}$$

and simplified to

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\sigma^2. \quad \text{(A.23)}$$

Alternately, this derivation can be arrived at by noting that the covariance of $\hat{\beta}$ in an unweighted regression is:

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{Z})^{-1}\sigma^2. \quad \text{(A.24)}$$

Substituting in the the starred variables yields:

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^{*t}\mathbf{Z}^*)^{-1}\sigma^2 \quad \text{(A.25)}$$

Returning to the original variables the equation becomes

$$\mathrm{Cov}(\beta) = [(\mathbf{P}^{-1}\mathbf{Z})^t\mathbf{P}^{-1}\mathbf{Z}]^{-1}\sigma^2 \quad \text{(A.26)}$$

which after the following matrix manipulations

$$\mathrm{Cov}(\beta) = [(\mathbf{Z}\mathbf{P}^{-1})^t\mathbf{P}^{-1}\mathbf{Z}]^{-1}\sigma^2 \quad \text{(A.27a)}$$

$$\mathrm{Cov}(\beta) = [\mathbf{Z}^t(\mathbf{P}^{-1})^t\mathbf{P}^{-1}\mathbf{Z}]^{-1}\sigma^2 \quad \text{(A.27b)}$$

$$\mathrm{Cov}(\beta) = (\mathbf{Z}^t\mathbf{P}^{-1}\mathbf{P}^{-1}\mathbf{Z})^{-1}\sigma^2 \quad \text{(A.27c)}$$

becomes

$$\mathrm{Cov}(\hat{\beta}) = (\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\sigma^2. \qquad \text{(A.28)}$$

The covariance for $\hat{\beta}$ from an unweighted regression is derived in Johnson and Wichern and the covariance of $\hat{\beta}$ for a weighted regression is given in Draper and Smith.

The variance, σ^2, is estimated by

$$s^2 = (\mathbf{y} - \mathbf{X}\hat{\beta})^t\mathbf{V}^{-1}(\mathbf{y} - \mathbf{X}\hat{\beta})/(n-p) \qquad \text{(A.29)}$$

where n is the number of data points and p is the number of estimated coefficients in the model.

APPENDIX B

DERIVATION OF PREDICTION CONFIDENCE INTERVAL FOR WEIGHTED REGRESSION

The prediction confidence interval for the linear model of this thesis, is derived in this appendix. The prediction confidence interval for the case where β is estimated through standard unweighted regression and error is assumed uniform at all points is derived in Johnson and Wichern. According to the model the response at the point z_0 is

$$y_0 = z_0^t\beta + \varepsilon_0 \tag{B.1}$$

which is estimated by

$$\hat{y}_0 = z_0^t\hat{\beta}. \tag{B.2}$$

Thus the error in the prediction is

$$y_0 - \hat{y}_0 = (z_0^t\beta + \varepsilon_0 - z_0^t\hat{\beta}) \tag{B.3}$$

We observe that since $\hat{\beta}$ is normally distributed, the linear combination $z_0^t\hat{\beta}$ is also normally distributed. Further, ε_0 has a normal distribution, so the quantity $y_0 - \hat{y}_0$ is normally distributed. Now we must find the mean and the variance of $y_0 - \hat{y}_0$. The expected value of $y_0 - \hat{y}_0$ is

$$E(y_0 - \hat{y}_0) = E(z_0^t\beta + \varepsilon_0 - z_0^t\hat{\beta}) = z_0^t\beta + 0 - z_0^t\beta = 0 \tag{B.4}$$

Since $z_0^t\beta$ is constant and $z_0^t\hat{\beta}$ and ε_0 are independent

$$var(y_0 - \hat{y}_0) = var(z_0^t\beta + \varepsilon_0 - z_0^t\hat{\beta}) = var(\varepsilon_0) + var(z_0^t\hat{\beta}) \tag{B.5}$$

At this point the derivation deviates from the standard derivation for an unweighted regression. The variance of ε_0 is

$$\text{var}(\varepsilon_0) = w_0\sigma^2 = \exp[(\mathbf{x}_0-\mathbf{x}_r)^t\mathbf{Q}(\mathbf{x}_0-\mathbf{x}_r)/(2m)]\sigma^2 \qquad \text{(B.6)}$$

and the variance of $\mathbf{z}_0^t\hat{\beta}$ is

$$\text{var}(\mathbf{z}_0^t\hat{\beta}) = \mathbf{z}_0^t\text{Cov}(\hat{\beta})\mathbf{z}_0 = \mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0\sigma^2 \qquad \text{(B.7)}$$

Substituting equations (B.6) and (B.7) into (B.5), the variance of $y_0 - \hat{y}_0$ can be written as

$$\text{var}(y_0-\hat{y}_0) = \mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0\sigma^2 + w_0\sigma^2 \qquad \text{(B.8)}$$

The first term represents the variance due to the error at point $\mathbf{x}_0$, and the second term represents the variance due to the error in estimating β. We can now write the distribution of $y_0 - \hat{y}_0$ as

$$y_0 - \hat{y}_0 \sim N(0,\mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0\sigma^2 + w_0\sigma^2) \qquad \text{(B.9)}$$

From this distribution of y, it can be shown that

$$(y_0-\hat{y}_0)/\sqrt{\mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0s^2 + w_0s^2} \sim t \qquad \text{(B.10)}$$

The confidence interval for the prediction value of y_0 is then derived in the usual manner yielding:

$$y_0 = \hat{y}_0 \pm t_{\alpha/2,}\sqrt{\mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0s^2 + w_0s^2} \qquad \text{(B.11)}$$

or

$$y_0 = \hat{y}_0 \pm t_{\alpha/2}\sqrt{\mathbf{z}_0^t(\mathbf{Z}^t\mathbf{V}^{-1}\mathbf{Z})^{-1}\mathbf{z}_0s^2+\exp[(\mathbf{x}_0 - \mathbf{x}_r)^t\mathbf{Q}(\mathbf{x}_0 - \mathbf{x}_r)/(2m)]s^2} \qquad \text{(B.12)}$$

The prediction confidence interval for the case where β is estimated through unweighted regression is derived in Johnson and Wichern.

APPENDIX C

BASIC OPTIMIZATION ALGORITHM

The optimization problem considered in this thesis is of the form

$$\max f(\mathbf{x})$$

subject to

$$\mathbf{g}(\mathbf{x}) \geq \mathbf{c} \qquad \mathbf{h}(\mathbf{x}) \geq \mathbf{b} \tag{C.1}$$

where

$\mathbf{h}(\mathbf{x})$ is the set of linear constraints.
$\mathbf{g}(\mathbf{x})$ is the set of nonlinear constraints.

In our algorithm there are only two nonlinear constraints. They which arise from the constraints which determine the feasible region, since the constraints that the user imposes are required to be linear.

The algorithm used to solve this problem is based on the gradient projection method. The algorithm is moves from the current feasible point $\mathbf{x}_k$ to an improved feasible point $\mathbf{x}_{k+1}$. The gradient projection method draws its name from the manner in which the search direction, $\mathbf{s}$, is determined. If the current point $\mathbf{x}_k$ is on the inside of the feasible region the search direction is the direction of the gradient of the objective function. If the current point is on the border of the feasible region, then the search direction is the projection of the gradient on the binding constraint(s). This results in the search for the optimum point moving along the binding constraints in the second case. The second case is illustrated in Figure 1. Active constraints are those for which the equality holds - i.e. $\mathbf{g}_i(\mathbf{x}_k) = \mathbf{c}_i$, and $\mathbf{h}_j(\mathbf{x}_k) = \mathbf{b}_j$. Binding constraints are those active constraints which restrict the direction of the optimization.

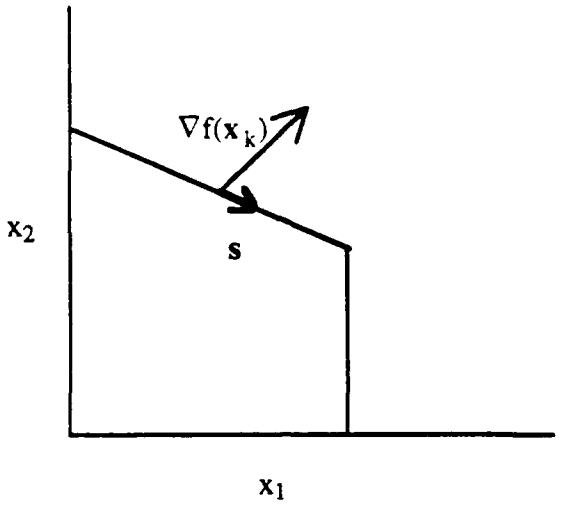

Figure 1: Projection of the gradient of the objective function on the binding constraint

The projection matrix is calculated using

$$\mathbf{P} = [\mathbf{I} - \mathbf{A}^t(\mathbf{A}\mathbf{A}^t)^{-1}\mathbf{A}], \tag{C.2}$$

where the **A** matrix consists of the coefficients of the binding constraints. Each binding constraint is a row in A. In order to determine which of the active constraints are binding the La Grange multipliers, λ, can be used. The multipliers are calculated according to

$$\lambda = \mathbf{A}_a^t(\mathbf{A}_a\mathbf{A}_a^t)^{-1}\nabla f(\mathbf{x}_k), \tag{C.3}$$

where $\mathbf{A}_a$ is the matrix of all active constraints with each active constraint occupying a row of $\mathbf{A}_a$. The positive elements of λ indicate that the corresponding constraints are binding. The search direction is then calculated as

$$\mathbf{s} = \mathbf{P}\nabla f. \tag{C.4}$$

The incorporation of the nonlinear constraints causes some difficulty. In order to incorporate the nonlinear constraints they are linearized to be the gradient of the constraint at the point of interest. The projection matrix is calculated using these linearized constraints. This, however, creates problems.

Let $y = x_k + \alpha s$ where x_k is a feasible point, s is a direction which violates none of the linear or linearized constraints, and α is the step length. Since the linearized constraints are only accurate at the point the point y will violate the nonlinear constraints. This is demonstrated in Figure 2.

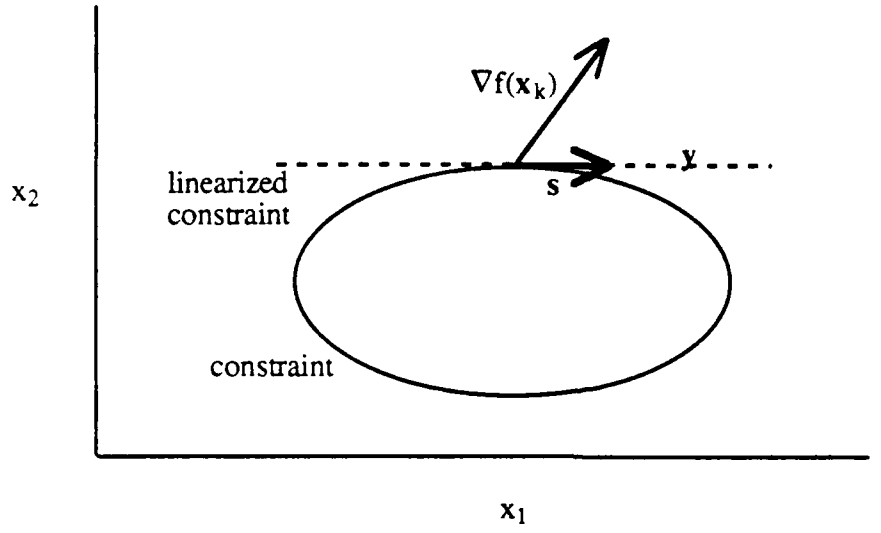

Figure 2: Projection of the gradient of the objective function on a linearized binding constraint

In order to return to the feasible region the the following iterate process is used.

$$y_{j+1} = y_j - A^t(AA^t)^{-1}[g_b(y_j) - c_b] \qquad C.5$$

where

$g_b(y_j)$ is the set of the all of the binding constraints evaluated at y_j and c_b is the set of corresponding boundary conditions. This process will never return the point exactly to the feasible region, but it can come to within δ of the feasible region. Let $w(y)$ refer to the results of the process for the infeasible point y.

Outline of the Algorithm

The basic idea of the optimization algorithm is to move from the current feasible point to a better point. This process of moving from point to point in the direction can be broken

down into two steps: 1) calculating the search direction and 2) determining how far to move in the search direction to the next point.

STEP 0

start at an initial feasible point x_0, and choose a maximum step for the nonlinear case. The nonlinear step size is denoted by α_n

STEP 1: Calculating the search direction **s**

The first step can be broken down into several parts:

1. Calculate $\nabla f(x_k)$, the gradient of $f(x_k)$.

2. Determine the active constraints, and find the binding ones from

$$\lambda = A_a^t(A_a A_a^t)^{-1}\nabla f(x_k) \quad , \lambda_i > 0$$

(Note that most algorithms don't differentiate between binding and nonbinding active constraints until **s** = 0. This is done for computational reasons.)

3. Use the binding constraints to create **A**. The projection matrix can then be calculated according to

$$\mathbf{P} = [\, \mathbf{I} - \mathbf{A}^t(\mathbf{A}\mathbf{A}^t)^{-1}\mathbf{A}]$$

(Note that if none of the constraints are binding **P = I,** the identity matrix.)

4. The search direction is then calculated from

$$\mathbf{s} = \mathbf{P}\nabla\mathbf{f}$$

If the absolute value of the search direction is zero then the constrained optimum has been reached. For practical purposes the optimum is considered to have been reached if the absolute value of **s** is sufficiently small.

5. if $|s| \leq \varepsilon$ then stop.

Step 2: determining how far to move in the search direction to the next point

The second step is broken down into two cases. The first case is when the nonlinear constraints are nonbinding. The second is when the nonlinear constraints are binding.

Case 1: The nonlinear constraints are not binding

1. First the maximum step size is determined.

Max $\{ \alpha_1 : \mathbf{u} = \mathbf{x} + \alpha_1 \mathbf{s}$ is feasible $\}$

2. The best point along this line is determined. This line search is done using a secant search. (This is a one dimensional search.) In the code for this thesis the algorithm stops after n_1 consecutive line searches.

Max $\{ \alpha_2 : f(\mathbf{x}_k + \alpha_2 \mathbf{s}), 0 \leq \alpha_2 \leq \alpha_1 \}$

3. The current point is then updated to be this point.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_2 \mathbf{s}$$

4. return to Step 1

Three examples of Step 1 and Case 1 of Step 2 are shown in Figures 3,4,5.

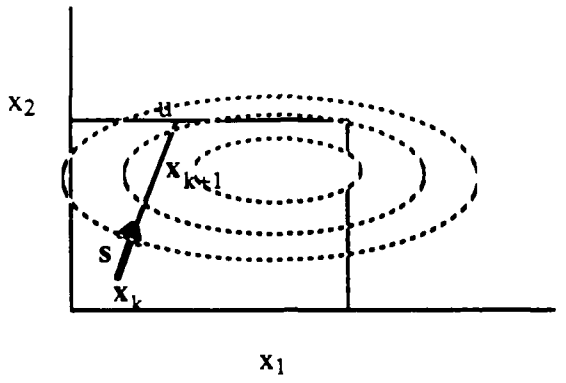

Figure 3: The search direction is the direction of the objective function gradient. The optimum point in this direction, x_{k+1}, is between the bound and the current point, x_k.

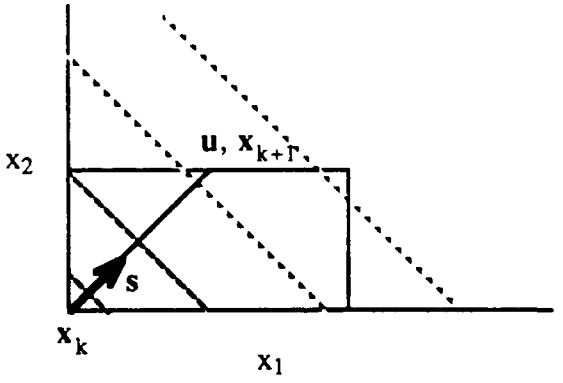

Figure 4: The search direction is the direction of the objective function gradient. There are two active constraints at the point x_k but neither is binding. The optimum point in this direction, x_{k+1}, is on the bound.

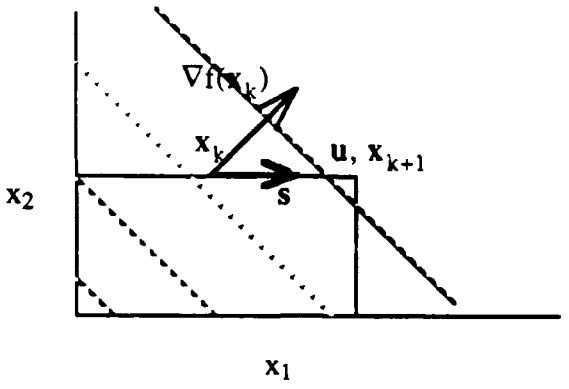

Figure 5: The search direction is the direction of the projection of objective function gradient on the binding constraint. The optimum point in this direction, x_{k+1}, is on the bound.

Case 2: at least one of the nonlinear constraints is binding

If the any of the nonlinear constraints are binding then a new procedure must be followed. This procedure consists of taking steps of size α_n along the nonlinear constraints as long as the step results in an improved point. The step size α_n is halved if the new point does not result in a better point.

1. Find a point where the projection results in a feasible solution. For the code written for this thesis, this is done by cutting the step length α_n in half until a point is found where the projection back on to the constraints results in a feasible point (see Figure 6).

Min {p: $w(y = x_k + (1/2)^p \alpha_n s)$ is feasible}

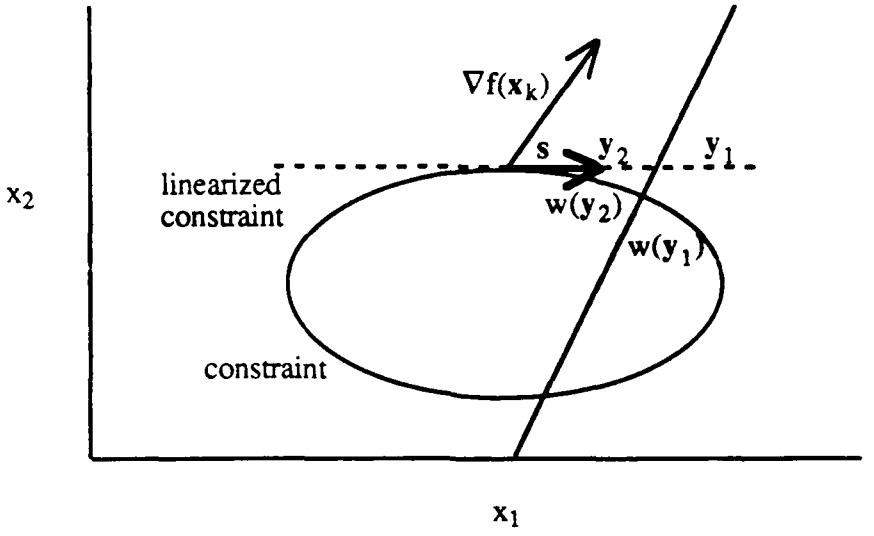

Figure 6: Halving step size to find a feasible projection on constraints

2. Next determine if this new point resulted in an increase in the objective function.

a) If it resulted in an increase, then update the current point to be this new point and the step size remains the same (see Figure 7). In order to speed the optimization process, the step size is doubled if seven consecutive steps at the same step size have resulted in improved points and doubling the step size will result in a step size less than the maximum step size for the nonlinear case set in Step 0.

If $f[w(y)] \geq f(x_k)$ then $\alpha_n^{k+1} = \alpha_n^k$ and $x_{k+1} = w(y)$.

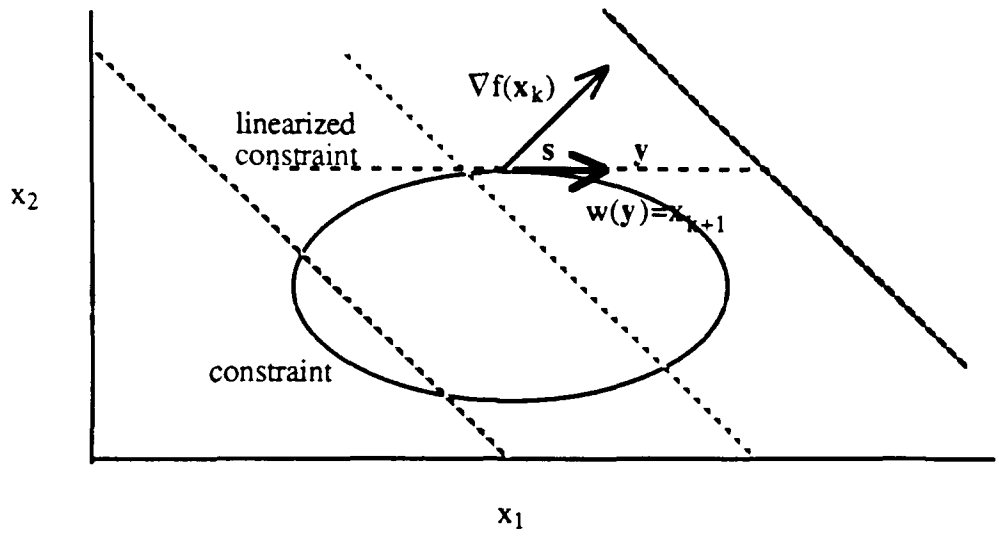

Figure 7: Current point is updated to be w(y), the point on the nonlinear constraint when y is a better point than $\mathbf{x}_k$

b) If the new point is an inferior point, the step size is cut in half and the current point remains the same.

$$\text{If } f[w(y)] < f(\mathbf{x}_k) \text{ , then } \alpha_n^{k+1} = 1/2\alpha_n^k \text{ and } \mathbf{x}_{k+1} = \mathbf{x}_k.$$

3. If $\alpha_n \le \varepsilon_2$ or $\alpha_n \le \varepsilon_3$, $f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \ge \varepsilon_4$ and the number of cycles is greater than n_2, then stop, otherwise return to step 1.

APPENDIX D

THE SEQUENTIAL OPTIMIZATION PROGRAM

```
#include <stdio.h>
#include <math.h>
/*****************************************************************/
/*                          global variables                     */
/*****************************************************************/
double v[50][50],k1,bc[20],con_mat[50][50],cur[20],aaa[50][50],sigma;
int pointer,ncon,flag,novar,ncoeff,pl,flag2,flag3,flag9;
double c[30],g[20],s[20],t,aa[50][50],sig2[20],ave[20],hv[10];
double var2[50][50],ave2[20],d;
/*****************************************************************/
main(argc,argv)
   int argc;
   char *argv[];
{
/*****************************************************************/
/*                    external variables in main                 */
/*****************************************************************/
extern double v[50][50],k1,bc[20],con_mat[50][50],cur[20],aaa[50][50],sigma;
extern int pointer,ncon,flag,novar,ncoeff,flag2;
extern double c[30],g[20],s[20],t,sig2[20],ave[20],hv[10];
/*****************************************************************/
/*                          functions in main                    */
/*****************************************************************/
double abss(),agrad(),bk(),bound(),create(),conck(),conse(),cpm(),grad();
double lines(),mm(),dconck(),dconse();
double modelg(),mvm(),nran(),reg(),trans(),vran(),w(),wbound();
/*****************************************************************/
/*                    declaration of variables in main           */
/*****************************************************************/
 FILE *sp,*sp3,*sp9;
 double  b[50][50],a[50][50],proj[20],slope,f1,f3,alphan;
 double y,at,alpha_old;
 double xtr[50][50];
 int check,i,j,l,nodat,k,r,p,timeold,repeat;
 double lv[10],ad,time, f[20],junk;
 double curm[20],ym,dck;
 static double st[32] = {
   {0,6.3,2.9,2.3,2.1,2.0,1.94,1.9,1.86,1.83,1.812,1.8,1.78,1.77,1.76,1.75,
   1.75,1.74,1.73,1.73,1.725,1.72,1.72,1.71,1.71,1.71,1.7,1.7,1.7,1.7,1.7,1.64},
 };
  static double chi[17]={0.0,2.71,4.61,6.25,7.78,9.24,10.64,12.02,13.36,14.68,
  15.99,17.28,18.55,19.81,21.06,22.31,23.54};
 double alpha,pm[50][50],sub();
 double ab,am,tck,yhat;

/*                          random variable seed is set          */
 srand(10);
/*****************************************************************/
/*                          beginning of algorithm               */
/*****************************************************************/
```

```
for(p=1;p<=25;++p)
 {

  if(p==1)
     {
  /*************************************************************************/
  /*              beginning of initialization routine                      */
  /*              this step is only executed before the first cycle        */
  /*             it asks the user to set the parameters such as the scaling */
  /*             constants and the initial reference value of y, K1= .1*yref */
  /*************************************************************************/

        printf("how many variables?\n");
        scanf("%d",&novar);
        printf("what is y ref\n");
        scanf("%f",&y);
        printf("what is # coefficients\n");
        scanf("%d",&ncoeff);
        t=pow(y*y,0.5)*0.1;
        for(j=1;j<=novar;++j)
          {
           printf("what are high and low values for variable %d?\n",j);
           scanf("%f %f",&hv[j],&lv[j]);
           hv[j]=hv[j]-lv[j];
          }

        sp3=fopen("numdat","r");
        fscanf(sp3,"%d",&nodat);
        fclose(sp3);

     }
  /*************************************************************************/
  /*            advice is given and the results of the experiment are collected */
  /*************************************************************************/
  if(p>=2)
     {
      vran(cur,curm);
      printf("run expt at \n");
      for(j=1;j<=novar;++j)
        {
         printf("variable %d = %f\n",j,cur[j]);
        }


/*              input new data and update numdat and exper.dat files          */

        printf("what is y?\n");
        scanf("%f",&y);
        ym=y/*+nran()*0.005;*/;
        printf("y measured = %f\n",ym);
        sp3=fopen("numdat","r");
        fscanf(sp3,"%d",&nodat);
        fclose(sp3);
```

```
        nodat=nodat+1;
        sp3=fopen("numdat","w");
        fprintf(sp3,"%d\n",nodat);
        fclose(sp3);
        t=0.1*pow(y*y,0.5);

/*              append new data to exper.dat file                              */

        printf("enter the data actually run\n");
        for(k=1;k<=novar;++k)
          {
           scanf("%f",&cur[k]);
        printf("got the data \n");
          }
        sp=fopen("exper.dat","a");
        fprintf(sp,"%f ",ym);

        for(k=1;k<=novar;++k)
          {
           fprintf(sp,"%f ",cur[k]);
          }
        fprintf(sp,"\n");
        fclose(sp);

/*************************************************************************/
/*                              book keeping                             */
/*************************************************************************/
        yhat=modelg(cur);
        tck=conck(cur);
        bk(yhat,p,tck,y);
       }

/*************************************************************************/
/*                      call  reg to update the coefficients             */
/*************************************************************************/
    reg(nodat,novar,hv,v,&sigma,g,c);
/********************* ***********************************************/
/*              calculate the k1 value (the appropriate t)               */
/*************************************************************************/
    d=2.0*chi[novar];
    r=nodat-ncoeff;
    if (r>=31)
        {
         r=31;
        }
    k1=st[r];

    j=0;
    for(l=1;l<=novar;++l)
        {
         cur[l]=g[l];
        }
```

```
    time=0.0;
    check=0;
    repeat=0;
    alpha_old=0.0;
    /*******************************************************/
    /*                    start of optimization                    */
    /*******************************************************/
    at =1.0;
    while (j<=0.0)
        {

/*      first the search direction, s, is determined        */

          grad(f);
          cpm(pm,f,time);
          mvm(s,pm,f,novar,novar);

/*      if |s| is small then the optimization is completed    */

          if(abss(s,novar)<=0.001)
           {
            if(time==0.0)
                {
                 j=1;
                 printf("help\n");
                 goto end;
                }
            j=1;
            tck=conck(cur);
            printf(" tck = %f\n",tck);
            dck=dconck(cur);
            printf(" dck = %f\n",dck);
           }
          else
           {

 /*   case where neither of the nonlinear constraints is binding      */

            if (flag==0 && flag3==0)
                {

/*        finds the boundary that will first be encountered     */
/*        by travelling in the direction of the s vector       */

                ab=bound(ncon,novar,s,cur,con_mat,bc,alpha);
                am =conse(xtr,g,novar,a,b,st,r,ncoeff,v,sigma,ab,s,t,cur);
                ad =dconse(xtr,g,novar,a,b,st,r,ncoeff,v,sigma,ab,s,t,cur);
                alpha=am;
                if(am>=ab)
                  {
                   alpha=ab;
                  }
                if(ad<alpha)
                  {
```

```
                    alpha=ad;
                   }

/*      if slope is positive at the bound then next point          */
/*      is point at the bound do line search in direction of       */
/*              the s vector                                       */

                   slope=agrad(alpha);
                   if(slope>0.0)
                    {
                     for (i=1;i<=novar;++i)
                          {
                           cur[i]=cur[i]+alpha*s[i];
                          }
                    }
                   else

/*  if slope is negative at bound then a secant line search is */
/* performed by the subroutine lines to find nest point       */

                   {
                    junk=lines(0.0,alpha,slope);
                    if(junk <= 0.000000001)
                         {

                          j=1;
                          tck=conck(cur);
                          printf(" tck = %f\n",tck);
                          dck=dconck(cur);
                          printf(" dck = %f\n",dck);
                         }
                    for (i=1;i<=novar;++i)
                         {
                          cur[i]=cur[i]+junk*s[i];
                         }
/* check # of line searches, quit if it is to large */

                    if(time=timeold+1)
                         {
                          repeat=repeat+1;
                         }
                    else
                         {
                          repeat=0;
                         }
                    if(repeat>=50)
                         {
                          j=1;
                          tck=conck(cur);
                          repeat=0;
                          printf("not repeating another line search");
                         }
                    timeold=time;
                   }
```

```
                }
            else

/*          the nonlinear constraints are  binding                  */

                {
                 alpha=wbound(at,proj);
                 if (alpha>=at) alpha=at;

/*      counting # of consecutive steps at a given size             */

                 if(alpha==alpha_old)
                   {
                    check=check+1;
                    if (check>=8) check=0;
                    if(check>=7 && alpha<=0.5)
                        {
                         alpha=2.0*alpha;
                         at=alpha;
                        }

                   }
                 else
                   {
                    alpha_old=alpha;
                    check=0;
                   }

/*          evaluating the model at old and new point               */

                 f1=modelg(cur);
                 f3=modelg(proj);
                 /*          printf(" f1 = %f, f3=%f\n",f1,f3);*/

/* update current point to be this new point if new point is better */

                 if(f3>=f1)
                   {
                    for (i=1;i<=novar;++i)
                        {
                         cur[i]=proj[i];
                        }
                    junk=f3-f1;

/*              check for stopping conditions                       */

                    if(at<=.001 && time>=1500 && junk<=0.0000001)
                        {
                         printf("we're taking the coward's route home");

                         tck=conck(cur);
                         printf(" tck = %f\n",tck);
                         dck=dconck(cur);
                         printf(" dck = %f\n",dck);
```

```
                        j=1;
                       }
                    }

/* if new point is better then halve step size          */
                   else
                    {
                     at=0.5*at;
                    }

/*  check for stopping conditions                       */

                   if (alpha <= 0.0000001)
                    {
                     j=1;
                     tck=conck(cur);
                     printf(" tck = %f\n",tck);
                     dck=dconck(cur);
                     printf(" dck = %f\n",dck);
                    }
                  }
              }
             time =time+1;
            }
    /* ************************************************************************/
    /*                                 end of optimizer                       */
    /* ********************************************************************** */
   }
 end: /* this is it */;
}
/*************************************************************************/
/*              end of main program and start of subroutines             */
/*************************************************************************/
/*                      subroutine to do matrix multiplication           */
/*                      a=x1*x2    x1 is nxp and x2 is pxm               */
/*************************************************************************/
double mm(a,x1,x2,n,p,m)
    int n,m,p;
    double a[50][50],x1[50][50],x2[50][50];
{
 int i,j,l;
 for (i=1;i<=n;++i)
   {
    for(j=1;j<=m;++j)
        {
         a[i][j]=0.0;
         for(l=1;l<=p;++l)
           {
            a[i][j]=a[i][j]+x1[i][l]*x2[l][j];
           }
        }
   }
 }
/*************************************************************************/
```

```
/*          subroutine to create predictor variables from control variable          */
/*******************************************************************/
double create(x,xr,nodat)
double xr[20],x[50][50];
int nodat;
{
 extern int flag2,flag9;
 int j,l,counter;
 double temp[20];
 for(j=1;j<=3;++j)
   {
    temp[j]=x[nodat][j];
   }

/* check to see if conversion will cause math error, if so flag2 = 0 */

if(temp[1]<0.0 || temp[3]<=0.0)
  {
   flag2=0;
   goto qu;
  }
 flag9=1;
x[nodat][1]=1.0;
x[nodat][2]=temp[1];
x[nodat][3]=temp[2];
x[nodat][4]=temp[3];
x[nodat][5]=temp[1]*temp[1];
x[nodat][6]=temp[2]*temp[2];
x[nodat][7]=temp[3]*temp[3];
x[nodat][8]=temp[1]*temp[2];
x[nodat][9]=temp[1]*temp[3];
x[nodat][10]=temp[2]*temp[3];

 qu:;
}*

/* *****************************************************************/
/*                        subroutine to transpose x-vector                        */
/*                  xt is the transpose of x, x is 1xm vector                     */
/*******************************************************************/
double trans(x,xt,p)
   double x[50][50],xt[50][50];
   int p;
{
 int m;
 for(m=1;m<=p;++m)
   {
    xt[m][1]=x[1][m];
   }
}
/* *****************************************************************/
/*                    subroutine to  multiply matrix and vector                   */
/*                       a = h*vec  h is nxp,  vec is px1                         */
/* *****************************************************************/
```

```
double mvm(a,h,vec,n,p)
    double a[20],h[50][50],vec[20];
    int n,p;
{
 int j,i;
 for (i=1;i<=n;++i)
   {
    a[i]=0.0;
    for (j=1;j<=p;++j)
        {
         a[i]=a[i]+h[i][j]*vec[j];
         }
   }
}
/* ************************************************************** */
/*                    gradient for prediction error constraint    */
/*                          is numerically determined             */
/* ************************************************************** */
double gradh(f,vec)
    double f[20],vec[20];

{

extern novar,ncoeff;
int l,i;
double conck(),xp[20],p[20];
for(l=0;l<=novar;++l)
 {
  xp[l]=vec[l]+.00000001;
  for(i=1;i<=l-1;++i)
    {
        p[i]=vec[i];
    }
  if(l>=1)
    {
        p[l]=xp[l];
    }
  for(i=l+1;i<=novar;++i)
    {
        p[i]=vec[i];
    }
  f[l]=conck(p);
  if (l>=1)
    {
        f[l]=(f[l]-f[0])/.00000001;
/*      printf("%f\n",f[l]);*/
    }
 }
}
/* ************************************************************** */
/*                    gradient for distance constraint            */
/*                      is numerically determined                 */
/* ************************************************************** */
double dgradh(f,vec)
```

```
    double f[20],vec[20];

{

extern novar,ncoeff;
int l,i;
double dconck(),xp[20],p[20];
for(l=0;l<=novar;++l)
  {
   xp[l]=vec[l]+.00000001;
   for(i=1;i<=l-1;++i)
     {
        p[i]=vec[i];
     }
   if(l>=1)
     {
        p[l]=xp[l];
     }
   for(i=l+1;i<=novar;++i)
     {
        p[i]=vec[i];
     }
   f[l]=dconck(p);
   if (l>=1)
     {
        f[l]=(f[l]-f[0])/.00000001;
/*      printf("%f \n",f[l]);*/
     }
  }
}
/* ********************************************************************** */
/*                                gradient for objective function         */
/*                                    is numerically determined           */
/* ********************************************************************** */
double grad(f)
    double f[20];
  {
extern double cur[20];
int l,i;
double modelg(),xp[20],p[20];
for(l=0;l<=novar;++l)
  {
   xp[l]=cur[l]+.00000001;
   for(i=1;i<=l-1;++i)
     {
        p[i]=cur[i];
     }
   if(l>=1)
     {
        p[l]=xp[l];
     }
   for(i=l+1;i<=novar;++i)
     {
        p[i]=cur[i];
```

```
    }
   f[l]=modelg(p);
   if (l>=1)
     {
        f[l]=(f[l]-f[0])/.00000001;
/*      printf("%f \n",f[l]);*/
     }
  }
}

/* ********************************************************** */
/*                        subroutine to create p matrix       */
/*                     the P matrix is a projection matrix    */
/* ********************************************************** */
double cpm(pm,f,time)
   double pm[50][50],f[20],time;
{
 extern double cur[20], con_mat[50][50],aaa[50][50],bc[20],g[20];
 extern int pointer,novar,coeff,flag,flag3;
 extern double aa[50][50],k1,t,d;
 FILE *sp2;
 int i,l,j,m,junk,count,info;
 double aat[50][50],a2[20],a[50][50],b[50][50],fh[20];
 double dgedi_(),dgefa_(),mvm(),ipvt[50],work[50],det[3],u[20];
 double conck(),dconck(),gradh(),dgradh();
 sp2=fopen("constr","r");
 fscanf(sp2,"%d",&ncon);

/* initializing the pm matrix by setting it equal to 0 */
for(i=1;i<=novar;++i)
   {
    for(l=1;l<=novar;++l)
        {
         pm[l][i]=0.0;
        }
   }

/*      read in user constraints from constr       */

 for (i=1;i<=ncon;++i)
   {
    for(l=1;l<=novar;++l)
        {
         fscanf(sp2,"%f",&con_mat[i][l]);
        }
    fscanf(sp2,"%f",&bc[i]);
   }
 fclose(sp2);

/*       set up aa matrix with active constraints       */

 mvm(a2,con_mat,cur,ncon,novar);
 pointer=0.0;
 for (i=1;i<=ncon;++i)
```

```
    {
     if(bc[i]-a2[i]>=-.01)
          {
           pointer=pointer+1;
           for(m=1;m<=novar;++m)
             {
              aa[pointer][m]=con_mat[i][m];
             }

          }
    }

/*    see if nonlinear prediction constraint is active    */

  if(conck(cur)>=t-.01)
    {
     if(time==0)
          {
           t=conck(cur);
           printf("changing t\n");
          }
     pointer=pointer+1;
     gradh(fh,cur);
     for(i=1;i<=novar;++i)
          {
           aa[pointer][i]=0.0 -fh[i];
          }
     flag=1;
    }
  else
    {
     flag=0;
    }
/* (flag and flag3 are global variables that indicate whether or not the*/
/* prediction error constraint and the distance constraint respectively are*/
/* binding)  */

/*  check to see if the nonlinear distance constraint is active        */

  if(dconck(cur)>=d-.01)
    {
     if(time==0)
          {
           d=dconck(cur);
           printf("changing d\n");
          }
     pointer=pointer+1;

/* if the constraint is active  linearize the nonlinear distance   */
/*        bound by using its gradient                              */

     dgradh(fh,cur);
     for(i=1;i<=novar;++i)
          {
```

```
        aa[pointer][i]=0.0 -fh[i];
       }
   flag3=1;
  }
else
  {
   flag3=0;
  }

for(j=1;j<=pointer;++j)
  {
   for(i=1;i<=novar;++i)
       {
        aat[i][j]=aa[j][i];
       }
  }

/*                              set up the identity matrix              */
if(pointer == 0)
  {
   for (i=1;i<=novar;++i)
       {
        pm[i][i]=1.0;
       }
  }
else
  {
   for (i=1;i<=novar;++i)
       {
        pm[i][i]=1.0;
       }
   mm(a,aa,aat,pointer,novar,pointer);
   /* invert matrix */
   for(i=1;i<=pointer;++i)
       {
        for(m=1;m<=pointer;++m)
          {
           a[i-1][m-1]=a[i][m];
          }
       }
   dgefa_(a,&pointer,&pointer,ipvt,&info);
   dgedi_(a,&pointer,&pointer,ipvt,det,work,01);
   for(i=pointer-1;i>=0;--i)
       {
        for(m=pointer-1;m>=0;--m)
          {
           a[i+1][m+1]=a[i][m];
          }
       }
   mm(b,a,aa,pointer,pointer,novar);

/*          check LaGrange multipliers to see which active constraints are binding          */

   mvm(u,b,f,pointer,novar);
```

```
    for(i=2;i<=pointer+1;++i)
        {
         count=0;
         for(l=i-1;l>=1;--l)
           {
            if(u[l]>0.0)
                {
                 count=count+1;
                 if(l==pointer-flag3 && flag==1)
                   {
                    flag=0;
                    printf("changed flag\n");
                   }
                 if(l==pointer && flag3==1)
                   {
                    flag3=0;
                    printf("changed flag3\n");
                   }

                }
           }
         if(count>0)
           {
            for(j=1;j<=novar;++j)
                {
                 aa[i-count][j]=aa[i][j];
                }
           }
        }

/*                  pointer is global variable and is # of binding constraints          */

    pointer=pointer-count;

/*                                        construct projection matrix                   */

    for(j=1;j<=pointer;++j)
        {
         for(i=1;i<=novar;++i)
           {
            aat[i][j]=aa[j][i];
           }
        }
    mm(a,aa,aat,pointer,novar,pointer);
    /* invert matrix */
    for(i=1;i<=pointer;++i)
        {
         for(m=1;m<=pointer;++m)
           {
            a[i-1][m-1]=a[i][m];
           }
        }
    dgefa_(a,&pointer,&pointer,ipvt,&info);
    dgedi_(a,&pointer,&pointer,ipvt,det,work,01);
```

```
    for(i=pointer-1;i>=0;--i)
        {
         for(m=pointer-1;m>=0;--m)
           {
            a[i+1][m+1]=a[i][m];
           }
        }
    mm(aaa,aat,a,novar,pointer,pointer);

    mm(a,aaa,aa,novar,pointer,novar);
    for(i=1;i<=novar;++i)
        {
         for(l=1;l<=novar;++l)
           {
            pm[i][l]=pm[i][l]-a[i][l];
           }
        }
   }

}

/* ************************************************** */
/*            alpha grad numerically calculates df/d(alpha)          */
/* ************************************************** */
double agrad(alpha)
   double alpha;

{

 int l,i;
 double modelg(),grad[3],result,xp[20],junk;
 for(l=0;l<=1;++l)
   {
    w(.00001*l+alpha,xp);

    grad[l]=modelg(xp);
   }
 result=(grad[1]-grad[0])/.00001;
/*  printf("%f ",result);*/
 return(result);
}

/* ************************************************** */
/*            bound finds the stepsize in the s direction            */
/*                 to the nearest (linear) user constraint           */
/* ************************************************** */
double bound(ncon,novar,s,g,h,bc,alpha)
   double h[50][50],s[20],g[20],bc[20],alpha;
   int novar,ncon;
{
 double mvm(),atx[20],ats[20],amax[20],fabs();
 int i;
 mvm(atx,h,g,ncon,novar);
 mvm(ats,h,s,ncon,novar);
```

```
/*  printf(" %f %f %f \n",s[1],s[2],s[3]);*/
  for(i=1;i<=ncon;++i)
    {

          if(ats[i]==0.0 || fabs(ats[i]) <= 1*pow(10.0,-7.0))
            {
             amax[i]=1000.0;

            }
          else if (bc[i]/ats[i]-atx[i]/ats[i]>0.0)
            {
             amax[i]=(bc[i]-atx[i])/ats[i];
            }
          else
            {
             amax[i]=1000.0;
            }
      }
  /* find alpha */
  alpha=1000.0;
  for(i=1;i<=ncon;++i)
    {
     if(amax[i]<=alpha)
          {
           alpha=amax[i];
          }
    }
  return(alpha);
}
/*************************************************************/
/*                conck evaluates 1/2 the width of the prediction          */
/*                              confidence interval at x                   */
/*************************************************************/
double conck(x)
    double x[20];

{
extern double g[20],v[50][50],sigma,k1,hv[10],var2[50][50],ave2[20];
extern int novar,ncoeff,nodat;
int i;
double xt[50][50],xtr[50][50],a[50][50],b[50][50],create(),trans(),mm(),standard();
double w,c[50][50],f;
for(i=1;i<=novar;++i)
  {
   xtr[1][i]=x[i];
  }
w=0.0;
for(i=1;i<=novar;++i)
  {
    w=w+pow((x[i]-g[i])/hv[i],2.0);
  }
w=w/novar/2.0;
if(w>=10.0)
  {
```

```
   w=10.0;
  }
w=exp(w);
/* test for distance */
for(i=1;i<=novar;++i)
  {
   c[i][1]=x[i]-ave2[i];
  }
mm(a,var2,c,novar,novar,1);
for(i=1;i<=novar;++i)
  {
   c[1][i]=c[i][1];
  }
mm(b,c,a,1,novar,1);
f=b[1][1];
create(xtr,g,1);
standard(xtr);
trans(xtr,xt,ncoeff);
mm(b,xtr,v,1,ncoeff,ncoeff);
mm(a,b,xt,1,ncoeff,1);

a[1][1]=pow((w+a[1][1])*sigma,.5)*k1;
/*a[1][1]=pow(a[1][1],.5)*k1;*/

return(a[1][1]);
}
/*****************************************************************/
/*                        dconck evaluates the distance at x                    */
/*****************************************************************/
double dconck(x)
   double x[20];

{
extern double g[20],v[50][50],sigma,k1,hv[10],var2[50][50],ave2[20];
extern int novar,ncoeff,nodat;
int i;
double xt[50][50],xtr[50][50],a[50][50],b[50][50],create(),trans(),mm(),standard();
double w,c[50][50],f;
/* test for distance */
for(i=1;i<=novar;++i)
  {
   c[i][1]=x[i]-ave2[i];
  }
mm(a,var2,c,novar,novar,1);
for(i=1;i<=novar;++i)
  {
   c[1][i]=c[i][1];
  }
mm(b,c,a,1,novar,1);
f=b[1][1];
return(f);
}

/* ********************************************************** */
```

```
/*              conse finds the maximum step size based on the          */
/*                          prediction error constraint                 */
/* ******************************************************** */
double conse(xtr,g,novar,a,b,st,r,ncoeff,v,sigma,alpha,s,t,cur)
    double alpha,xtr[50][50],g[20],a[50][50],b[50][50],st[32],sigma;
    double v[50][50],s[20],t,cur[20];
    int novar,r,ncoeff;
{
 extern double hv[10];
 int k,j,i;
 double alphau,alphal,x1[50][50],x2[50][50],xt[50][50],w;
 double create(),standard();
 for(k=1;k<=novar;++k)
   {
    x1[1][k]=cur[k]+alpha*s[k];
   }

 for(k=1;k<=novar;++k)
   {
    x2[1][k]=cur[k];

   }

 j=0;
 alphau = alpha;
 alphal=0.0;

 /* if the calculated variance of y-predicted is not less than maximum */
 /* allowable value iterate to find a point equal to the maximum      */

 while(j<=0)
   {

    alpha=0.5*(alphau+alphal);
    for(k=1;k<=novar;++k)
        {
         xtr[1][k]=cur[k]+s[k]*alpha;
        }
    w=0.0;
    for(i=1;i<=novar;++i)
        {
         w=w+pow((xtr[1][i]-g[i])/hv[i],2.0);
        }
    w=w/2.0/novar;
    if(w>=10.0)
        {
         w=10.0;
        }
    w=exp(w);
    create(xtr,g,1);
    standard(xtr);
    trans(xtr,xt,ncoeff);
    mm(b,xtr,v,1,ncoeff,ncoeff);
    mm(a,b,xt,1,ncoeff,1);
```

```
    a[1][1]=pow((w+a[1][1])*sigma,.5)*st[r];
    if(a[1][1]>=t+.0001)
        {
         for(k=1;k<=novar;++k)
           {

            alphau=alpha;
           }
        }
    if(a[1][1]<=t-.0001)
        {
         for(k=1;k<=novar;++k)
           {
            alphal=alpha;

           }
        }
    /* condition for exiting the while loop */
    if(a[1][1]<=t+.0001 && a[1][1]>=t-.0001)
        {
         for(k=1;k<=novar;++k)
           {
            xtr[1][k]=xtr[1][k];
           }
         j=1;
        }
    if(alphau-alphal<=.00001) j=1;
   }
 return(alpha);
}
/* ********************************************************** */
/*              dconse finds the maximum step size based on the          */
/*                                  distance constraint                  */
/* ********************************************************** */

double dconse(xtr,g,novar,a,b,st,r,ncoeff,v,sigma,alpha,s,t,cur)
    double alpha,xtr[50][50],g[20],a[50][50],b[50][50],st[32],sigma;
    double v[50][50],s[20],t,cur[20];
    int novar,r,ncoeff;
{
 extern double var2[50][50],ave2[20],d;
 int k,j,i;
 double alphau,alphal,x1[50][50],x2[50][50],xt[50][50],w;
 double create(),standard();
 for(k=1;k<=novar;++k)
   {
    x1[1][k]=cur[k]+alpha*s[k];
   }

 for(k=1;k<=novar;++k)
   {
    x2[1][k]=cur[k];

   }
```

```
j=0;
alphau = alpha;
alphal=0.0;
while(j<=0)
  {

    alpha=0.5*(alphau+alphal);
    for(k=1;k<=novar;++k)
        {
         xtr[1][k]=cur[k]+s[k]*alpha-ave2[k];
        }
    trans(xtr,xt,novar);
     mm(b,xtr,var2,1,novar,novar);
     mm(a,b,xt,1,novar,1);
    if(a[1][1]>=d+.0001)
        {
         for(k=1;k<=novar;++k)
           {

            alphau=alpha;
           }
        }
    if(a[1][1]<=d-.0001)
        {
         for(k=1;k<=novar;++k)
           {
            alphal=alpha;

           }
        }
    /* condition for exiting the while loop */
    if(a[1][1]<=d+.0001 && a[1][1]>=d-.0001)
        {
         for(k=1;k<=novar;++k)
           {
            xtr[1][k]=xtr[1][k];
           }
         j=1;
        }
    if(alphau-alphal<=.00001) j=1;
  }
 return(alpha);
}

/* ************************************************************************/
/*            lines is a line search for max value on a line between two bounds      */
/*                              this is done with secant search                      */
/* ************************************************************************/
double lines(l,temp,slope)
double  temp,l,slope;
{
int j;
double bound1,bound2,q,abs(),z,agrad(),d,e,p,rgrad,lgrad;
```

```
j=0;
bound1=temp;
bound2=l;
rgrad=slope;
lgrad=agrad(l);
if(lgrad<=0.0)
  {
   temp =0.0;
   goto out;
  }
while(j==0)
  {
d=rgrad;
e=rgrad-lgrad;
q=(temp-l);
z=(temp-d/e*q);
if(z < bound2 || z > bound1)
  {
   printf("there is a mistake\n");
  }
p=agrad(z);
/*printf("this is p,z,l,r %f %f %f %f\n",p,z,l,temp);*/
if(temp -l<= 0.00001)
  {
   j=1;
  }

if(p<=.000001 && p>=0.0)
  {
   j=1;
  }
else
  {
   if(p>=0.0)
     {
         l=z;
         lgrad=p;
     }
   else
     {
         temp=z;
         rgrad=p;
     }

  }
}
temp=z;
 out:;
return(temp);
}

/*****************************************************************/
/*                    modelg evaluates the model at x            */
/*****************************************************************/
```

```
double modelg(x)
   double  x[20];

{
 extern double c[30],g[20];
 extern int ncoeff,novar;
 double y,pow(),big[50][50],standard();
 int i;
  /*create evaluate model */
 y=0.0;
 for(i=1;i<=novar;++i)
   {
   big[1][i]=x[i];
   }

 create(big,g,1);
 standard(big);
 for(i=1;i<=ncoeff;++i)
   {
    y=y+c[i]*(big[1][i]);
   }
 return(y);
}
/* ************************************************ */
/*            abss determines the absolute value of s            */
/*  ************************************************ */
double abss(s,novar)
   double s[20];
   int novar;
{
 double pow(),result;
 int i;
 result=0.0;
 for (i=1;i<=novar;++i)
   {
    result=pow(s[i],2.0)+result;
   }
 result=pow(result,0.5);
 return(result);
}
/*************************************************************/
/*                        bk prints results into file                        */
/*************************************************************/
double bk(yhat,p,tck,y)
   double yhat,tck,y;
   int p;
{
double up,down;
FILE *sp;
up=yhat+tck;
down=yhat-tck;
sp=fopen("tp","a");
fprintf(sp," %d %f %f %f %f\n",p-1,up,yhat,down,y);
fclose(sp);
```

```
}
/*****************************************************************/
/*                    this subroutine does weighted regression                    */
/*****************************************************************/
double reg(nodat,novar,xs,var,sigma,xr,c)
   int nodat,novar;
   double var[50][50],xr[20],c[30],xs[10],*sigma;
{
 extern int ncoeff,flag9;
 extern double ave[20],sig2[20],ave2[20],var2[50][50];
 double  mm(),y[50][50],v[50][50],xn[50],fr,pt[5],x[50][50],xt[50][50];
 double fabs(),ipvt[50],det[3],work[50],a[50][50],b[50][50],g;
 double exp(),pow(),proj[50][50],ave1[20],proj2[50][50];
 int k,l,info,counter,j,i,junk;
 double dgefa_(),dgedi_(),create(),z[50][50],s[50][50];
 FILE *sp;
 /* ************************************************************  */
 /*              beginning of program to do local regression                  */
 /*                                                                           */
 /*****************************************************************/

 /* read in data       */
 sp=fopen("exper.dat","r");
 printf("%d %d \n",nodat,novar);
 /* create x  and xt matrix */
 for(i=1;i<=nodat-1;++i)
   {
    fscanf(sp,"%f",&y[i][1]);
    for(j=1;j<=novar;++j)
        {
         fscanf(sp,"%f",&x[i][j]);
        }
   }
 fscanf(sp,"%f",&fr);
 for(j=1;j<=novar;++j)
   {
    fscanf(sp,"%f",&xr[j]);
   }
 fclose(sp);
 y[nodat][1]=fr;
 for(j=1;j<=novar;++j)
   {
   x[nodat][j]=xr[j];
   }

/* for each data pt find weight and its contribution to sum of squares */
/*  based on normalized distance from reference point            */

 for(i=1;i<=nodat-1;++i)
   {
    for(j=1;j<=nodat;++j)
        {
         v[i][j]=0.0;
        }
```

```
    }
  for(i=1;i<=nodat;++i)
   {
    xn[i]=0.0;
    for(j=1;j<=novar;++j)
        {
         xn[i]=xn[i]+pow((x[i][j]-xr[j])/xs[j],2.0);
        }
    v[i][i]=exp(-xn[i]/2.0/novar):
   }
/*************************************************************************/
/*                        this part is for distance constraint           */
/*************************************************************************/

/*                              S = {X'[I - 1/n(11')]X}/(n - 1)           */
/*                                where 1is a vector of ones              */
for(i=1;i<=nodat;++i)
 {
  for(j=1;j<=nodat;++j)
   {
       proj2[i][j]=0.0-1.0/(nodat);
       if(i==j) proj2[i][j]=1.0-1.0/(nodat);
   }
 }
 for(i=1;i<=novar;++i)
  {
   for(j=1;j<=nodat;++j)
        {
         xt[i][j]=x[j][i];
        }
  }
 mm(a,xt,proj2,novar,nodat,nodat);
 mm(var2,a,x,novar,nodat,novar);
 for(j=1;j<=novar;++j)
  {
   for(i=1;i<=novar;++i)
        {
         var2[i][j]=var2[i][j]/(nodat-1.0);
        }
  }
 for(i=1;i<=novar;++i)
  {
   for(j=1;j<=novar;++j)
        {
         var2[i-1][j-1]=var2[i][j];
        }
  }
 junk=novar;
 dgefa_(var2,&junk,&junk,ipvt,&info);
 dgedi_(var2,&junk,&junk,ipvt,det,work,11);
 for(i=ncoeff;i>=0;--i)
  {
   for(j=ncoeff;j>=0;--j)
        {
```

```
            var2[i+1][j+1]=var2[i][j];
          }
    }
/*                                find mean vector of X                                */
 for(i=1;i<=novar;++i)
    {
     ave2[i]=0.0;
    }
 for(j=1;j<=nodat;++j)
    {
     for(i=1;i<=novar;++i)
          {
           ave2[i]=ave2[i]+x[j][i];
          }
    }
 for(i=1;i<=novar;++i)
    {
     ave2[i]=ave2[i]/(nodat);
    }

 for(i=1;i<=nodat;++i)
    {
     create(x,xr,i);
    }
/* create projection matrix = identity matrix */
for(i=1;i<=nodat;++i)
  {
   for(j=1;j<=nodat;++j)
     {
          proj[i][j]= 0.0;
          if(i==j) proj[i][j]=1.0;

     }
  }

/*                          find mean vector of predictor variables                   */
 for(i=1;i<=ncoeff;++i)

    ave[i]=0.0;
   }
 for(j=1;j<=nodat;++j)
   {
    for(i=1;i<=ncoeff;++i)
          {
           ave[i]=ave[i]+x[j][i];
          }
   }
 for(i=1;i<=ncoeff;++i)
   {
    ave[i]=ave[i]/(nodat);
   }
 mm(z,proj,x,nodat,nodat,ncoeff);

 /*************************************************************************/
```

```
/*                        create the x transpose (xt) matrix                    */
/**************************************************************************/
for(i=1;i<=ncoeff;++i)
  {
   for(j=1;j<=nodat;++j)
        {
         xt[i][j]=z[j][i];
        }
  }

/*                    scale predictor variables by their average value          */
/*                      (this is done to aid in the matrix inversion)           */

 for(i=1;i<=nodat;++i)
  {
   for(j=1;j<=ncoeff;++j)
        {
         z[i][j]=z[i][j]/ave[j];
         xt[j][i]=z[i][j];
        }
  }

 /** ***********************************************************************/
 /*                          b=(xt*v*x)**-1*(xt*v*y)                         */
 /**************************************************************************/
 mm(b,xt,v,ncoeff,nodat,nodat);
 mm(a,b,z,ncoeff,nodat,ncoeff);

/* copy a */
 for(i=1;i<=ncoeff;++i)
  {
   for(j=1;j<=ncoeff;++j)
        {
         var[i][j]=a[i][j];
        }
  }
 /* find inverse */
 for(i=1;i<=ncoeff;++i)
  {
   for(j=1;j<=ncoeff;++j)
        {
         a[i-1][j-1]=a[i][j];
        }
  }
 junk=ncoeff;
 dgefa_(a,&junk,&junk,ipvt,&info);
 dgedi_(a,&junk,&junk,ipvt,det,work,11);
 for(i=ncoeff;i>=0;--i)
  {
   for(j=ncoeff;j>=0;--j)
        {
         a[i+1][j+1]=a[i][j];
        }
  }
```

```
 mm(b,a,var,ncoeff,ncoeff,ncoeff);
/* ************************************************************** */
/*          calculate X'V**-1X for use in prediction error calculations          */
/*************************************************************************/
for(i=1;i<=ncoeff;++i)
  {
   for(j=1;j<=ncoeff;++j)
        {
         var[i][j]=a[i][j];
        }
  }
 mm(b,a,xt,ncoeff,ncoeff,nodat);
 mm(a,b,v,ncoeff,nodat,nodat);
 mm(b,a,y,ncoeff,nodat,1);
 printf("the coefficients are : ");
 c[0]=fr;
 for(i=1;i<=ncoeff;++i)
  {
   printf(" %f \n",b[i][1]);
   c[i]=b[i][1];
   printf("beta revised =  %f\n",c[i]/ave[i]);
  }
 printf("\n");
 /* ************************************************************ */
 /*                          find weighted variance                          */
 /*************************************************************/

 /* find est of y */
 mm(a,z,b,nodat,ncoeff,1);
/*  sp=fopen("resid","w");*/
 for(k=1;k<=nodat;++k)
  {
   z[k][1]=y[k][1]-a[k][1];
/*     printf("%f\n",z[k][1]);

    fprintf(sp,"%d %f\n",k,z[k][1]);*/

   xt[1][k]=z[k][1];
  }
 fclose(sp);
 mm(a,xt,v,1,nodat,nodat);
 mm(b,a,z,1,nodat,1);
 b[1][1]=b[1][1]/(nodat-ncoeff);
 *sigma=b[1][1];
 printf("sigma= %f\n",b[1][1]);
/*  for(i=1;i<=ncoeff;++i)
   {
    for(j=1;j<=ncoeff;++j)
        {
         var[i][j]=b[1][1]*var[i][j];
        }
   }
*/
}
```

```
/*********************************************************/
/*                nran generates normal random variables              */
/*********************************************************/
double nran()
{
 int srand(),rand();
 double z1,r1,r2,z2;


 r1= rand()/(pow(2.0,31.)-1.);
 r2=rand()/(pow(2.,31.)-1.);
/*  printf("r1=%f r2= %f ",r1,r2);*/
 z1=pow(-2.0*log(r1),0.5)*cos(2.0*3.14159*r2);
 z2=pow(-2.0*log(r1),0.5)*sin(2.0*3.14159*r2);
 /* printf("%f %f\n",z1,z2);*/
 return(z1);
}*
/*************************************************/
/*              vran can superimpose a random variable         */
/*                   on each of the control variables            */
/*************************************************/
double vran(cur,curm)
   double cur[20],curm[20];

{
 double nran();
 int i;
 curm[1]=nran();
 printf("rand variable = %f\n",curm[1]);
 for(i=1;i<=5;++i)
 {
  curm[i]=cur[i];
 }
}
/*************************************************************************/
/*                        find projection on to the nonlinear constraints                */
/*************************************************************************/
double w(alpha,proj)
   double alpha,proj[20];


   /* find the projection of the point on to the nonlinear constraint */
{
 extern double k1,cur[20],s[20],t,aa[50][50],d;
 extern int pointer,novar,ncoeff,flag,pl,flag2,flag3;
 double f[20],a[50][50],aat[50][50],aaa[50][50],h[20],proj2[20];
 int i,j,k,m,info,count;
 double tk,conck(),ipvt[50],work[50],det[3],gradh(),dck,dconck(),tck;
 for(i=0;i<=pointer;++i)
  {
   h[i]=0.0;
  }
 for(i=1;i<=novar;++i)
```

```
   {
    proj[i]=cur[i]+alpha*s[i];
   }
count=0;
if(flag3==0 && flag==0) goto q;
 wieder:;
 j=0;
 count=count+1;
 if (count>500)
   {
    flag2=0;
    goto q;
   }
 if (flag==1)
   {
    h[pointer-flag3]=t-conck(proj);
   }
 if (flag3==1)
   {
    h[pointer]=d-dconck(proj);
   }
 for(i=pointer-flag-flag3+1;i<=pointer;++i)
   {
    if(h[i]<=0.0-0.001)
        {
         j=1;
        }

   }
 if(j==0) goto q;

 /* case where the first constraint is binding          */
 if(flag==1)
   {


    if (flag2==0) goto q;
    j=0;
    /* update projecting matrix */
    gradh(f,proj);
    for(i=1;i<=novar;++i)
        {
         aa[pointer-flag3][i]=0.0-f[i];
        }
   }

/*     case where the second constraint is binding          */
 if(flag3==1)
   {
    h[pointer]=d-dconck(proj);
    if (flag2==0) goto q;
    j=0;
    /* update projecting matrix */
    dgradh(f,proj);
```

```
    for(i=1;i<=novar;++i)
        {
         aa[pointer][i]=0.0-f[i];
        }
   }
  for(k=1;k<=pointer;++k)
   {
    for(i=1;i<=novar;++i)
        {
         aat[i][k]=aa[k][i];
        }
   }

  mm(a,aa,aat,pointer,novar,pointer);
  /* invert matrix */
  for(i=1;i<=pointer;++i)
   {
    for(m=1;m<=pointer;++m)
        {
         a[i-1][m-1]=a[i][m];
        }
   }
  if (a[0][0]==0.0)
   {
    flag2=0;
    goto q;
   }
  dgefa_(a,&pointer,&pointer,ipvt,&info);
  dgedi_(a,&pointer,&pointer,ipvt,det,work,01);
  for(i=pointer-1;i>=0;--i)
   {
    for(m=pointer-1;m>=0;--m)
        {
         a[i+1][m+1]=a[i][m];
        }
   }
  mm(aaa,aat,a,novar,pointer,pointer);

  mvm(proj2,aaa,h,novar,pointer);
  for(i=1;i<=novar;++i)
   {
    proj[i]=proj[i]-proj2[i];
   }
goto wieder;
tk=conck(proj);
/*        printf("this is w end tk = %f\n",tk);*/
 q:;

}
/*****************************************************************/
/*                    wbound returns a feasible step size for the case          */
/*            where at least one of the nonlinear constraints is binding        */
/*****************************************************************/
double wbound(step,proj)
```

```
   double step,proj[20];
{
 extern double bc[20],con_mat[50][50],s[20],cur[20],t,d;
 extern int novar,ncon,flag2;
 int i,j;
 double u[20],alphau,alphal,alpha,w(),conck(),tck,dconck(),dck;
 alpha=step;

/* check to see if maximum nonlinear step size results in a feasible */
/*          projection on to the nonlinear constraints          */

 alphau=bound(ncon,novar,s,cur,con_mat,bc,alpha);
 if(alphau>=step)
   {
    alphau=step;
/*printf("were changing alphau\n");*/
   }
 alphal=0.0;
again:;
flag2=1;
 w(alphau,proj);

/*      check to see if this pt violates any constraints        */

 mvm(u,con_mat,proj,ncon,novar);
 j=1;
 for(i=1;i<=ncon;++i)
   {
    if(u[i]-bc[i]<=-.01)
        {
         j=0;
        }
    if(d-dconck(proj)<0.0-.01) j=0;
    if(t-conck(proj)<0.0-.01) j=0;
   }

/*       if the point is feasible then stop              */

  if (j==1 && flag2==1)
  {
   alpha =alphau;
   goto quit;
  }
  else

/*       if the point is not feasible, have the step size     */
    {
        printf("were in a new case\n");
        alphau =0.5*alphau;
        goto again;
    }

quit:;
 return(alpha);
```

```
}
/*********************************************************/
/*                standard divides the predictor variables */
/*                              by their average          */
/*********************************************************/
double standard(z)
   double z[50][50];
{
extern int ncoeff;
extern double ave[20],sig2[20];
int i;
for(i=1;i<=ncoeff;++i)
  {
  z[1][i]=z[1][i]/ave[i];
  }
}
```